KATEDRA PRŮMYSLOVÉHO INŽENÝRSTVÍ A MANAGEMENTU

DISERTAČNÍ PRÁCE

Plzeň, 2012

Ing. Pavel Raška

ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA STROJNÍ

Studijní program: P2301 Strojní inženýrství

Studijní obor: 2301V007 Průmyslové inženýrství a management

DISERTAČNÍ PRÁCE

OPTIMALIZAČNÍ METODY PRO DISKRÉTNÍ SIMULACI VÝROBNÍCH SYSTÉMŮ A VÝROBNÍCH PROCESŮ VE STROJÍRENSTVÍ

Autor: **Ing. Pavel Raška**

Vedoucí práce: **doc. Ing. Václav Votava, CSc.**

Akademický rok 2012/2013

Prohlášení o autorství

Předkládám tímto k posouzení a obhajobě disertační práci, zpracovanou na závěr studia na Fakultě strojní Západočeské univerzity v Plzni.

Prohlašuji, že jsem předloženou disertační práci na téma:

Optimalizační metody pro diskrétní simulaci výrobních systémů a výrobních procesů ve strojírenství

vypracoval samostatně, pod odborným dohledem školitele a za použití odborné literatury a pramenů uvedených v přiložené bibliografii.

V Plzni dne 1.11.2012 .....

Ing. Pavel Raška

Upozornění

Podle Zákona o právu autorském. č.35/1965 Sb. (175/1996 Sb. ČR) § 17 a Zákona o vysokých školách č. 111/1998 Sb. je využití a společenské uplatnění výsledků disertační práce, včetně uváděných vědeckých a technických poznatků nebo jakékoliv nakládání s nimi možné pouze na základě autorské smlouvy za souhlasu autora a Fakulty strojní Západočeské univerzity v Plzni.

Poděkování

Rád bych poděkoval svému školiteli doc. Ing. Václavu Votavovi, CSc. a kolegovi doc. Ing. Zdeňku Ulrychovi, PhD. za odborné vedení a cenné podněty, které pomohly vzniku této disertační práce. Děkuji také ostatním členům Katedry průmyslového inženýrství a managementu Západočeské univerzity v Plzni, jejichž náměty a připomínky přispěly k rozvoji této práce. Velké poděkování patří obětavé lásce mé rodiny a všech blízkých.

V Plzni dne 1.11.2012 .....

Ing. Pavel Raška

ANOTAČNÍ LIST DISERTAČNÍ PRÁCE

AUTOR	Ing. Raška			Pavel			
STUDIJNÍ OBOR	Průmyslové inženýrství a management						
VEDOUCÍ PRÁCE	doc. Ing. Votava, CSc.			Václav			
PRACOVIŠTĚ	ZČU - FST - KPV						
DRUH PRÁCE	**DISERTAČNÍ**						
NÁZEV PRÁCE	Optimalizační metody pro diskrétní simulaci výrobních systémů a výrobních procesů ve strojírenství						
FAKULTA	strojní		**KATEDRA**	KPV		**ROK ODEVZD.**	2012

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	247		**TEXTOVÁ ČÁST**	169		**GRAFICKÁ ČÁST**	78

STRUČNÝ POPIS	Tato disertační práce se zabývá využitím vhodných optimalizačních metod globální optimalizace v oblasti diskrétní simulace výrobních systémů a výrobních procesů. V práci byl zmapován současný stav v oblasti simulační optimalizace včetně specifikace jejích základních prvků a možných problémů spojených s globální optimalizací. Simulační experimentování na vybraných typech diskrétních simulačních modelů bylo prováděno pomocí vhodných optimalizačních metod (Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution a Evolution Strategy). Tyto optimalizační metody musely být modifikovány pro diskrétní simulační optimalizaci. Pro účely simulačního experimentování byla vytvořena vlastní softwarová aplikace, která obsahuje simulační optimalizátor pro řízení simulačních experimentů s cílem zjistit vhodné nastavení vstupních parametrů diskrétního simulačního modelu na základě specifikované účelové funkce. Softwarová aplikace také obsahuje experimentální základnu pro analýzu chování implementovaných modifikovaných optimalizačních algoritmů v závislosti na nastavených parametrech optimalizačních algoritmů. Pomocí experimentální základny byly provedeny hromadné optimalizační experimenty na vybraných typech diskrétních simulačních modelů s cílem vyhodnotit vhodnost optimalizačních algoritmů a určit vhodné nastavení parametrů optimalizačních algoritmů v závislosti na průběhu účelové funkce. Pro hodnocení chování optimalizačních algoritmů na simulačních modelech (účelových funkcích) byla specifikována metodika, která zahrnuje různé pohledy vyjádřené pomocí jednotlivých kritérií včetně vizuálního ohodnocení.
KLÍČOVÁ SLOVA	Diskrétní simulace výrobních systémů, globální optimalizace, optimalizace pomocí simulace, optimalizační algoritmy, Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution, Evolution Strategy.

SUMMARY SHEET

AUTHOR		Ing. Raška			Pavel		
FIELD OF STUDY		Industrial Engineering and Management					
SUPERVISOR		doc. Ing. Votava, CSc.			Václav		
INSTITUTION		ZČU - FST - KPV					
TYPE OF WORK		**Ph.D. THESIS**					
TITLE		Optimization Methods Used for Discrete Event Simulation of Production Systems and Production Processes					
FACULTY	Mechanical Engineering		**DEPARTURE**	KPV		**SUBMITTED IN**	2012

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	247		**TEXT PART**	169		**GRAPHICAL PART**	78

BRIEF DESCRIPTION	This thesis deals with the use of global optimization methods suitable for discrete simulation of manufacturing systems and production processes. The scope of this thesis is to analyse the current knowledge on simulation optimization, optimization of basic elements and possible problems related to global optimization. Simulation model experimentation was carried out using a selection of suitable optimization methods - Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution and Evolution Strategy. These optimization methods had to be modified for simulation optimization. A software application was developed for simulation model experimentation. The application consists of a simulation optimizer for managing the simulation experiments. The aim of the simulation optimizer is to determine the appropriate setting of the discrete event simulation model input parameters based on a specified objective function. The software application consists of an experimental base for analysis of the behaviour of the implemented modified optimization algorithms in relation to the settings of the optimization algorithm parameters. Many optimization experiments were performed on discrete event simulation models with the use of the designed experimental base. The goal of the experimentation was to evaluate the appropriateness of the optimization algorithms and determine the appropriate optimization algorithm parameter settings in relation to the behaviour of the objective function. The methodology for evaluating the behaviour of the optimization algorithms on simulation models (behaviour of objective function) was specified. This methodology includes different views expressed through different criteria including visual evaluation.
KEY WORDS	Discrete Event Simulation of Production Systems, Global Optimization, Simulation Optimization, Optimization Algorithms, Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution, Evolution Strategy.

KURZFASSUNG

AUTOR	Ing. Raška	Pavel	
STUDIENFACH	Industrielle Engineering und Management		
ARBEITSLEITER	doc. Ing. Votava, CSc.	Václav	
INSTITUTION	ZČU - FST - KPV		
ARBEITSTYPE	**DISSERTATION**		
TITEL	Optimierung Methoden für die diskrete Simulation der Produktionssystemen und Produktionsprozessen in dem Maschinenbau		

FAKULTAT	Maschinen		**KATHEDER**	KPV		**ABGEGEBEN IN**	2012

SEITENZAHL (A4 und aq. A4)

TOTAL	247		**TEXTTEILE**	169		**ANLAGE**	78

KURZBESCHREIBUNG	Diese Dissertationsarbeit behandelt die Ausnutzung von Optimierungsmethoden der globalen Optimierung im Gebiet der diskreten Simulation der Produktionssysteme und Produktionsprozesse. Der gleichzeitige Zustand der Aufgaben der Simulationsoptimierung inklusive der Spezifikation von Grundelementen und möglichen Problemen verbundenen mit der globalen Optimierung wurde dargestellt. Simulationsexperimentieren auf ausgewählten Typen der diskreten Simulationsmodelle wurde mit Hilfe der passenden Optimierungsmethoden (Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution a Evolution Strategy) gemacht. Diese Optimierungsmethoden wurden für die diskrete Simulationsoptimierung modifiziert. Zwecks der Simulationsoptimierung wurde eine eigene Softwareapplikation vorgeschlagen und hergestellt, die ein Simulationsoptimierungsglied für die Steuerung von Simulationsexperimenten enthält, damit eine passende Einstellung von Eingangsparametern des diskreten Simulationsmodelles aufgrund der Optimierung von spezifizierter Zweckfunktion festgestellt wurde. Die Softwareapplikation beinhaltet auch eine experimentale Basis für die Analyse der Verhaltung von implementierten modifizierten Algorithmen in der Abhängigkeit von ihren eingestellten Parametern. Mit Hilfe der experimentalen Plattform wurden massive Optimierungsexperimente auf ausgewählten Typen der diskreten Simulationsmodelle gemacht, damit die Zweckfunktion der Optimierungsalgorithmen bewertet wurde und passende Einstellung von Parametern der Optimierungsalgorithmen in der Abhängigkeit vom Verlauf der Zweckfunktion bestimmt wurde. Für die Bewertung der Behandlung von Optimierungsmethoden auf Simulationsmodellen wurde eine Methodik spezifiziert, die verschiedene, mit eigenen Kriterien inklusive visueller Bewertung ausgedrückte Ansichten beinhaltet.
SCHLÜSSELWÖRTER	Diskrete Simulation der Produktionssystemen, globale Optimierung, Optimierungsalgorithmen, Random Search, Hill Climbing, Tabu Search, Local Search, Downhill Simplex, Simulated Annealing, Differential Evolution, Evolution Strategy, Optimierung mit der Simulation.

Obsah

Seznam obrázků ... 13
Seznam tabulek ... 15
Seznam algoritmů ... 17
Použité značení ... 18
1 Úvod ... 20
2 Simulační optimalizace ... 22
3 Současný stav problematiky ... 24
3.1 Vstupy simulačního modelu - rozhodovací proměnné ... 24
3.2 Výstupy simulačního modelu - odezvy, účelová funkce ... 27
3.3 Omezení ... 32
3.3.1 Omezení vztažená na rozhodovací proměnné ... 33
3.3.2 Omezení vztažená na účelovou funkci ... 33
3.3.3 Metody práce s omezujícími podmínkami ... 35
3.3.3.1 Trest smrti (Death Penalty) ... 35
3.3.3.2 Penalizační funkce (Penalty Function) ... 37
3.3.3.3 Opravné algoritmy (Repair Algorithms) a speciální operátory ... 38
3.3.3.4 Dekodéry (Decoders) ... 41
3.4 Seznam ... 42
3.5 Globální optimalizace ... 43
3.6 Problémy globální optimalizace ... 46
3.6.1 NP-problém ... 46
3.6.2 Konvergence k lokálnímu optimu ... 47
3.6.3 Průběh účelové funkce ... 48
3.7 Obecné prvky globální optimalizace ... 49
3.7.1 Minimalizace (maximalizace) ... 49
3.7.2 Iterace ... 51
3.7.3 Kritérium ukončení ... 57
3.7.1 Množina optim ... 59
3.7.1.1 Aktualizace množiny optim ... 60
3.7.1.2 Extrakce optimálních prvků ... 61
3.7.1.3 Oříznutí množiny optim ... 62
4 Současný stav - globální optimalizační algoritmy ... 63
4.1 Formulace podmínky optimalizační úlohy ... 64
4.2 Stochastické algoritmy ... 65
4.2.1 Náhodné prohledávání (*Random Search*) ... 65
4.3 Deterministické algoritmy ... 66
4.3.1 Downhill Simplex ... 67
4.4 Gradientní metody ... 68

4.5 Pseudo-gradientní metody ... 68
4.5.1 Stochastický horolezecký algoritmus (*Stochastic Hill Climbing*) ... 69
4.5.2 Stochastické zakázané prohledávání (*Stochastic Tabu Search*) ... 70
4.5.3 Stochastické simulované žíhání (*Stochastic Simulated Annealing*) ... 71
4.5.4 Stochastické lokální prohledávání (*Stochastic Local Search*) ... 73
4.6 Evoluční výpočetní techniky ... 74
4.6.1 Populace v evolučních algoritmech ... 76
4.6.2 Reprodukce ... 79
4.6.3 Přiřazení fitness ... 79
4.6.4 Selekce ... 81
4.6.4.1 Zkrácený výběr (Truncation Selection) ... 82
4.6.4.2 Náhodná selekce (Random Selection) ... 83
4.6.4.3 Selekce přímo úměrná fitness (Fitness Proportionate Selection) ... 83
4.6.4.3.1 Selekce pomocí ruletového mechanismu ... 84
4.6.4.4 Turnajová selekce (Tournament Selection) ... 85
4.6.4.5 Selekce na základě uspořádání (Ordered Selection) ... 86
4.7 Evoluční strategie (*Evolution Strategy*) ... 87
4.8 Diferenciální evoluce (*Differential Evolution*) ... 89
5 Teoretická východiska z hodnocení současného stavu ... 93
6 Cíle disertační práce ... 94
7 Použité vědecké metody zkoumání ... 95
8 Vývoj softwarové aplikace simulační optimalizace ... 97
8.1 Implementované optimalizační metody ... 98
8.1.1 Náhodné prohledávání ... 98
8.1.2 Downhill Simplex ... 98
8.1.3 Stochastický horolezecký algoritmus ... 98
8.1.4 Stochastické zakázané prohledávání ... 98
8.1.5 Stochastické simulované žíhání ... 98
8.1.6 Stochastické lokální prohledávání ... 99
8.1.7 Evoluční strategie ... 99
8.1.8 Diferenciální evoluce ... 99
8.2 Popis prostředí softwarové aplikace simulační optimalizace ... 101
8.2.1 Vstupní parametry ... 101
8.2.2 Nastavení optimalizace ... 102
8.2.3 Simulační experimenty ... 104
8.2.4 Vyhodnocení simulačních experimentů ... 105
9 Simulační modely ... 110
9.1 „De Jong“ ... 110
9.2 „Rosenbrock“ ... 110
9.3 „Michalewicz“ ... 110

9.4 „Ackley“ ... 111
9.5 „Doprava“ ... 111
9.6 „Penalizace“ ... 111
9.7 „VyrobniLinka“ ... 112
10 Experimentování ... 113
10.1 Série – formulace podmínek ... 113
10.2 Časová náročnost a vzdálené řízení experimentů ... 115
11 Metodika hodnocení optimalizačních algoritmů podle různých kritérií ... 116
11.1 Kritérium - f1 - Počet nalezených optim nebo hodnot blízkých optimu ... 116
11.2 Kritérium - f2 - Rozdíl lokálního extrému od optima ... 125
11.3 Kritérium - f3 - Vzdálenost kvartilů ... 127
11.4 Kritérium - f4 - Rychlost nalezení optima ... 129
11.5 Kritérium - f5 – Konvergence ... 131
12 Metodika výběru vhodného nastavení parametrů optimalizačního algoritmu ... 134
12.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model . 134
12.2 Nalezení vhodného nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 135
12.3 Nalezení vhodného nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 136
13 Vhodné nastavení parametrů jednotlivých optimalizačních algoritmů na základě provedených sérií ... 137
13.1 Random Search ... 137
13.1.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 137
13.2 Downhill Simplex ... 137
13.2.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 138
13.2.1.1 Reflexe ... 138
13.2.1.2 Expanze ... 138
13.2.1.3 Kontrakce ... 139
13.2.1.4 Redukce ... 139
13.2.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 139
13.2.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 140
13.3 Hill Climbing ... 140
13.3.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 140
13.3.1.1 Rozptyl ... 141
13.3.1.2 Velikost populace ... 142
13.3.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 142
13.3.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 143
13.4 Tabu Search ... 144
13.4.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 144

13.4.1.1 Rozptyl ... 144
13.4.1.2 Velikost populace ... 144
13.4.1.3 Délka zakázaného seznamu ... 145
13.4.1.4 Optimalizace podle poslední populace ... 146
13.4.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 146
13.4.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 147
13.5 Simulated Annealing ... 147
13.5.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 147
13.5.1.1 Rozptyl ... 147
13.5.1.2 Měnit jen jeden parametr ... 148
13.5.1.3 Beta ... 149
13.5.1.4 Minimální teplota ... 150
13.5.1.5 Generovat dle rozptylu ... 151
13.5.1.6 Snižovat teplotu jen při přijetí horšího řešení ... 152
13.5.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 152
13.5.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 152
13.6 Local Search ... 153
13.6.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 153
13.6.1.1 Rozptyl ... 153
13.6.1 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 154
13.6.2 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 154
13.7 Evolution Strategy ... 154
13.7.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 155
13.7.1.1 Velikost populace ... 155
13.7.1.2 Mmp – počet potomků ... 156
13.7.1.3 q – počet předcházejících úspěchů ... 156
13.7.1.4 k – počet jedinců v turnaji ... 157
13.7.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 157
13.7.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 158
13.8 Differential Evolution ... 158
13.8.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model 159
13.8.1.1 Velikost populace ... 159
13.8.1.2 Koef_F – koeficient adaptivního pravidla ... 159
13.8.1.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných ... 160
13.8.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely ... 161

13.8.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely 161
14 Přínosy disertační práce 162
14.1 Teoretické přínosy práce 162
14.2 Praktické přínosy práce 163
15 Doporučení pro další postup 164
16 Závěr 165
17 Použitá literatura 166
18 Přehled publikační činnosti 168
18.1 Publikované práce 168
18.2 Řešené projekty 169
18.3 e-Book 169
18.4 Publikovaný software 169

Seznam obrázků

Obr. 1-1 Klasický přístup k řešení optimalizačních problémů za pomoci lidského faktoru 21
Obr. 2-1 Simulační optimalizace 22
Obr. 3-1 Dvourozměrný prohledávaný prostor 26
Obr. 3-2 Vyhodnocení výstupů ze simulačního modelu 28
Obr. 3-3 Příklad grafu účelové funkce – zobrazení 30
Obr. 3-4 Příklad metody váženého součtu 31
Obr. 3-5 Příklad grafu účelové funkce s omezením účelové funkce 35
Obr. 3-6 Příklady různého druhu omezení 38
Obr. 3-7 Příklad principu perturbace prvku 40
Obr. 3-8 Příklad transformace mezi prohledávaným prostorem a novým prohledávaným prostorem [13] 42
Obr. 3-9 Příklad možných výsledků globální optimalizace účelové funkce s omezením prostoru všech řešení na prohledávaný prostor, sousedství prvku 46
Obr. 3-10 Příklad předčasné konvergence optimalizačního algoritmu v prohledávaném prostoru [15] 47
Obr. 3-11 Příklady různých typů povrchu účelové funkce – hledání globálního minima účelové funkce 49
Obr. 3-12 Transformace hledání maxima účelové funkce na úlohu hledání minima účelové funkce 51
Obr. 3-13 Příklad průběhu optimalizačního algoritmu - iterace algoritmu 56
Obr. 3-14 Příklad množiny optim 59
Obr. 4-1 Princip Reflexe, Expanze, Kontrakce, Redukce 67
Obr. 4-2 Zacyklení stochastického horolezeckého algoritmu 69
Obr. 4-3 Pravděpodobnost přijmutí řešení u metody simulovaného žíhání [19] 73
Obr. 4-4 Základní cyklus evolučních algoritmů [15] 76
Obr. 4-5 Příklad generování nového jedince pomocí postupu RAND 90
Obr. 8-1 Funkce aplikace simulační optimalizace 97
Obr. 8-2 Princip členění vzhledem k počtu prováděných simulačních experimentů 105
Obr. 8-3 Příklad grafu průběhu hledání optima - simulační model výrobní linky (maximalizace hodnot účelové funkce) - simulační experimenty provedené pro konkrétní nastavení optimalizačního algoritmu 106
Obr. 8-4 Příklad výsledků jedné série (jedné krabice) pomocí charakteristik krabicového grafu na základě hodnot účelové funkce 107
Obr. 8-5 Příklad výsledků v krabicovém grafu poskytnutých optimalizačním algoritmem „Evoluční strategie“ – jedna série (jedno konkrétní nastavení) je zobrazena jako jedna krabice v grafu – simulační model výrobní linky - maximalizace účelové funkce 108
Obr. 8-6 Počet provedených simulačních experimentů do nalezení optima (nalezené řešení není globálním optimem) při 1. optimalizačním experimentu s konkrétním nastavením optimalizačního algoritmu v první sérii – maximalizace účelové funkce 109
Obr. 11-1 Průměrná úspěšnost algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu 118
Obr. 11-2 Průměrná úspěšnost algoritmů při nalezení optima pro jednotlivé simulační modely a všechny nastavené parametry algoritmu 118
Obr. 11-3 Absolutní úspěšnost sérií algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu 120
Obr. 11-4 Absolutní úspěšnost sérií algoritmů na jednotlivých modelech na základě nalezení optima 122
Obr. 11-5 Průměrná úspěšnost optimalizačních algoritmů na simulačním modelu „Penalizace“ 122
Obr. 11-6 Průběh účelové funkce modelu „Penalizace“ - umístění výchozího bodu, umístění globálního minima 123
Obr. 11-7 Průběh účelové funkce modelu „Penalizace“ - výřez oblasti účelové funkce okolo globálního minima 124
Obr. 11-8 Absolutní neúspěšnost algoritmů při nalezení optima 124

Obr. 11-9 Absolutní neúspěšnost sérií algoritmů na jednotlivých modelech při nalezení optima 125
Obr. 11-10 Průměr hodnot f2 (rozdíl lokálního extrému od optima) u absolutně neúspěšných sérií pro všechny simulační modely 126
Obr. 11-11 Průměr z hodnot f2 (rozdíl lokálního extrému od optima) u absolutně neúspěšných sérií..... 127
Obr. 11-12 Průměr z hodnot f3 (vzdálenost kvartilů) u všech sérií kromě absolutně úspěšných sérií pro všechny simulační modely 129
Obr. 11-13 Průměr z hodnot f3 (vzdálenost kvartilů) u všech sérií kromě absolutně úspěšných sérií 129
Obr. 11-14 Průměr z hodnot f4 (rychlost nalezení optima) pro jednotlivé algoritmy 130
Obr. 11-15 Průměr z hodnot f4 (rychlost nalezení optima) u absolutně úspěšných sérií..... 131
Obr. 11-16 Průměr z hodnot f5 (konvergence) pro jednotlivé algoritmy u absolutně úspěšných sérií..... 132
Obr. 11-17 Průměr z hodnot f5 (konvergence) u absolutně úspěšných sérií 133
Obr. 12-1 Stanovené penalizace (váhy) u jednotlivých kritérií 135
Obr. 13-1 Krabicový graf zachycující vliv výše hodnoty parametru „Beta“ a „Minimální teplota“ na hodnoty účelové funkce prvků u provedených sérií na simulačním modelu „De Jong“ - Simulated Annealing 149
Obr. 13-2 Vliv opětovného zvýšení teploty (zvýšení pravděpodobnosti přijetí horšího řešení) na průběh optimalizačního experimentu u simulačního modelu "De Jong"- Simulated Annealing..... 150

Seznam tabulek

Tabulka 2-1 Ukázka optimalizačního software na trhu 23
Tabulka 8-1 Seznam implementovaných optimalizačních algoritmů a jejich parametrů 101
Tabulka 8-2 Charakteristiky krabicového grafu vztahující se k výsledkům poskytnutých optimalizačním algoritmem Evoluční strategie v 1. sérii – tj. při určitém nastavení parametrů optimalizačního algoritmu – simulační model výrobní linky - maximalizace účelové funkce 107
Tabulka 10-1 Provedené série na simulačních modelech 114
Tabulka 11-1 Provedené série optimalizačního algoritmu Downhill Simplex s $VelikostKroku = 1 \cdot 10 - 5$ na simulačním modelu „DeJong“ 121
Tabulka 11-2 Váhy pro jednotlivé rozsahy u hodnot účelové funkce poskytnutých výsledků jedné série v případě minimalizace účelové funkce 128
Tabulka 13-1 Doporučené hodnoty u parametru „Reflexe“ - Downhill Simplex 138
Tabulka 13-2 Doporučené hodnoty u parametru „Expanze“ - Downhill Simplex 139
Tabulka 13-3 Doporučené hodnoty u parametru „Kontrakce“- Downhill Simplex 139
Tabulka 13-4 Doporučené hodnoty u parametru „Redukce“- Downhill Simplex 139
Tabulka 13-5 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Downhill Simplex 140
Tabulka 13-6 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Downhill Simplex 140
Tabulka 13-7 Testované hodnoty parametrů - Hill Climbing 141
Tabulka 13-8 Doporučené hodnoty u parametru „Rozptyl“ - algoritmus Hill Climbing 142
Tabulka 13-9 Doporučené hodnoty u parametru „Velikost populace“ - algoritmus Hill Climbing 142
Tabulka 13-10 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely – Hill Climbing 143
Tabulka 13-11 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Hill Climbing 143
Tabulka 13-12 Doporučené hodnoty u parametru „Rozptyl“ - Tabu Search 144
Tabulka 13-13 Doporučené hodnoty u parametru „Velikost populace“ - algoritmus Tabu Search 145
Tabulka 13-14 Provedené série dalších hodnot parametru „Tabu Length“ na všech simulačních modelech ... 145
Tabulka 13-15 Doporučené hodnoty u parametru „Tabu Length“ - Tabu Search 146
Tabulka 13-16 Doporučené hodnoty u parametru „Optimalizace podle poslední populace“ - Tabu Search 146
Tabulka 13-17 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Tabu Search 146
Tabulka 13-18 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Tabu Search 147
Tabulka 13-19 Doporučené hodnoty u parametru „Rozptyl“ - Simulated Annealing 148
Tabulka 13-20 Testované hodnoty parametrů - Simulated Annealing 148
Tabulka 13-21 Doporučené hodnoty u parametru „Měnit jen jeden parametr“ - Simulated Annealing 149
Tabulka 13-22 Doporučené hodnoty u parametru „Beta“ - Simulated Annealing 150
Tabulka 13-23 Doporučené hodnoty u parametru „Minimální teplota“ - Simulated Annealing 151
Tabulka 13-24 Doporučené hodnoty u parametru „Generovat dle rozptylu“ - Simulated Annealing 151
Tabulka 13-25 Doporučené hodnoty u parametru „Snižovat teplotu jen při přijetí horšího řešení“ - Simulated Annealing 152
Tabulka 13-26 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Simulated Annealing 152
Tabulka 13-27 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Simulated Annealing 153

Tabulka 13-28 Doporučené hodnoty u parametru „Rozptyl“ - Local Search 154
Tabulka 13-29 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Local Search..... 154
Tabulka 13-30 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Local Search 154
Tabulka 13-31 Doporučené hodnoty u parametru „Velikost populace“ – Evolution Strategy 156
Tabulka 13-32 Doporučené hodnoty u parametru „Mmp“ - Evolution Strategy..... 156
Tabulka 13-33 Doporučené hodnoty u parametru „q“ - Evolution Strategy 157
Tabulka 13-34 Doporučené hodnoty u parametru „k“ - Evolution Strategy..... 157
Tabulka 13-35 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Evolution Strategy..... 158
Tabulka 13-36 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Evolution Strategy..... 158
Tabulka 13-37 Doporučené hodnoty u parametru „VelikostPopulace“ – Differential Evolution 159
Tabulka 13-38 Doporučené hodnoty u parametru „Koef_F“ - Differential Evolution..... 160
Tabulka 13-39 Doporučené hodnotyu parametru „Koef_C“ - Differential Evolution 161
Tabulka 13-40 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Differential Evolution 161
Tabulka 13-41 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Differential Evolution 161

Seznam algoritmů

Algoritmus 3-1 Generování náhodného čísla v rozsahu mezí na základě rovnoměrného rozdělení - Randomu 36
Algoritmus 3-2 Transformace souřadnic prvku mimo meze 39
Algoritmus 3-3 Perturbace souřadnic prvku mimo meze [14] 41
Algoritmus 3-4 Komparace hodnot účelové funkce dvou prvků v případě minimalizace účelové funkce 50
Algoritmus 3-5 Generování prvku - Create 52
Algoritmus 3-6 Generování množiny prvků - CreatePop 53
Algoritmus 3-7 Transformace složek prvku na základě rovnoměrného rozdělení - Mutate$u$ 54
Algoritmus 3-8 Transformace složek prvku na základě normálního rozdělení - Mutate$n$ 55
Algoritmus 3-9 Příklad iterace a kritéria ukončení 57
Algoritmus 3-10 UpdateOptimalSet [10] 61
Algoritmus 3-11 ExtractOptimalSet 62
Algoritmus 4-1 NonElitist [10] 78
Algoritmus 4-2 Elitist [10] 78
Algoritmus 4-3 Komparace hodnot fitness funkce dvou prvků v případě minimalizace fitness funkce 82
Algoritmus 11-1 AssignFitnessFindingOptimum 117
Algoritmus 11-2 SeriesSuccess 119
Algoritmus 11-3 AssignFitnessBest 126

Použité značení

X … Prostor všech řešení.

$\tilde{X}$ … Prohledávaný prostor optimalizační metodou.

$\tilde{X}_j$ … Prohledávaný interval j-té rozhodovací proměnné.

j… Index j-té rozhodovací proměnné.

n… Dimenze prostoru všech rozhodovacích proměnných.

a_j … Dolní mez prohledávaného j-tého intervalu.

b_j … Horní mez prohledávaného j-tého intervalu.

$\mathbf{X}$ … Prvek (vektor) obsahující hodnoty jednotlivých rozhodovacích proměnných.

x_j … Hodnota souřadnice bodu – hodnota j-té rozhodovací proměnné prvku $\mathbf{X}$, tj., $\mathbf{X}[j] = x_j \forall j: j = \{0,1,2,\dots,n-1\}$.

$\mathbf{X}_i$ … i-tý prvek.

i… Index prvku.

${}_{(i)}x_j$ … Hodnota j-té rozhodovací proměnné X_j. Geometricky ${}_{(i)}x_j$ znázorňuje souřadnici i-tého prvku na ose X_j v prohledávaném prostoru. Jedná se o složku prvku - vektoru $\mathbf{X}_i$ (generovaného v i-té iteraci) v n-rozměrném prohledávaném prostoru $\tilde{\mathrm{X}}$.

e … Délka okolí sousedství bodu (prvku) jednotlivé rozhodovací proměnné.

m … Počet vygenerovaných bodů v jednom kole algoritmu – počet iterací (v kontextu evolučních algoritmů - počet vygenerovaných jedinců v populaci).

O … Množina odezev, tj., $O = \{o_0(\mathbf{X}), o_1(\mathbf{X}), \dots, o_{q_o-1}(\mathbf{X})\}$.

$o_0(\mathbf{X})$ … První odezva ze simulačního modelu.

q_o … Počet odezev simulačního modelu.

$o_{q_o-1}(\mathbf{X})$… Poslední odezva simulačního modelu.

$F(\mathbf{X})$… Účelová funkce – obor hodnot jsou reálná čísla, tj., $F(\mathbf{X}) \subseteq \mathbb{R}$.

y … Hodnota účelové funkce, tj., $y = F(O(\mathbf{X}))$.

$\mathbb{R}$ … množina reálných čísel.

$\mathbb{N}$ … množina přirozených čísel.

F_l … l-tá účelová funkce, tj., $F_l \in \zeta, l = \{0,1,2,\dots,q-1\}$.

l … Index účelové funkce.

F_0 … První účelová funkce.

q … Počet účelových funkcí.

$X_{Feasible}$ … Prostor přípustných řešení, $X_{Feasible} \subseteq \tilde{X} \subseteq X$.

$X_{Non-feasible}$ … Prostor nepřípustných řešení, $X_{Non-feasible} = X \setminus X_{Feasible}$.

$g(\mathbf{X})$ … Funkce omezení kladené na rozhodovací proměnnou.

$g\big(F(\mathbf{X})\big)$ … Funkce omezení kladené na účelovou funkci.

$F_{\mathrm{cost}}(\mathbf{X})$ … penalizační funkce

$\widehat{\mathbf{X}}_1$ … Lokální maximum.

$\widehat{\mathbf{X}}$ … Globální maximum.

$\breve{\mathbf{X}}_1$ … Lokální minimum.

$\breve{\mathbf{X}}$ … Globální minimum.

$\mathbf{X}_1^*$ … Lokální optimum účelové funkce.

$\mathbf{X}^*$ … Globální optimum účelové funkce.

X^* … Množina optim.

m^* … Počet prvků množiny optim.

${}^{(0)}\mathbf{X}_i$ … Počáteční přípustné řešení.

k … Index kola algoritmu.

${}^{(k)}\mathbf{X}_i$ …Vygenerovaný prvek $\mathbf{X}_i$ v k-tém kole algoritmu (v kontextu evolučních algoritmů – jedinec $\mathbf{X}_i$ vygenerovaný v k-té populaci)

$\mathbf{X}_i^*$ ... Kandidát řešení $\mathbf{X}_i^* \in X^*$. Možné nalezené řešení.

${}^{(k)}\mathbf{X}_i^*$ … Nejlepší prvek v k-tém kole algoritmu (v kontextu evolučních algoritmů – nejlepší jedinec k-té populace).

$F\left({}^{(k)}\mathbf{X}_i\right)$ … Hodnota účelové funkce prvku $\mathbf{X}_i$ v k-tém kole algoritmu (v kontextu evolučních algoritmů – hodnota účelové funkce jedince $\mathbf{X}_i$ v k-té populaci).

1 Úvod

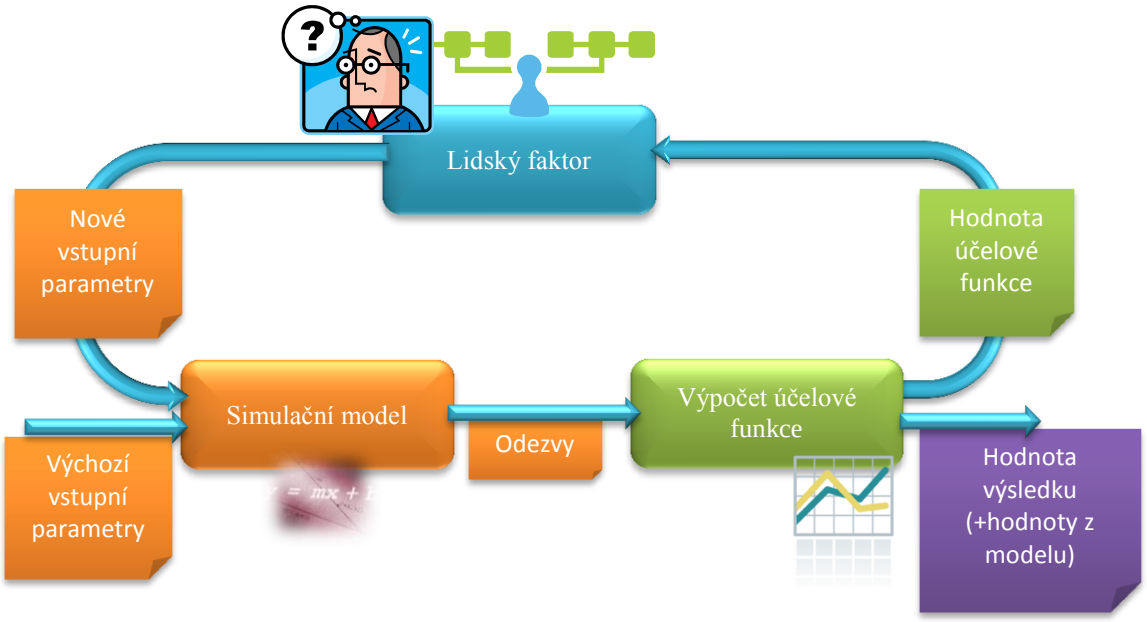
Téměř v každém dnešním oboru se setkáváme s problémem zvaným „volba toho správného“. Velmi často jsme v průmyslové praxi omezováni různými vnitřními faktory podniku (např. zdroje) a vnějším okolím podniku - dodavatelé, odběratelé, stát apod. Můžeme říci, že skoro ve všech případech je snaha vybrat nejlepší variantu ze všech možných přípustných variant na základě určitého cíle, tedy optimalizovat.

Poměrně dlouhou dobu byla optimalizace řešena pomocí klasického matematického aparátu nebo lidského (intuitivního) rozhodování na základě zjednodušeného vnímání nastalé situace. Problém ale nastává v situaci, kdy potřebujeme nalézt uspokojivá (někdy nejlepší) řešení určitého problému, který nemá jednoduchý charakter.

V oblasti výrobních systémů a výrobních procesů a jejich simulace (na které je tato disertační práce zaměřena) existuje řada omezujících vnitropodnikových faktorů, např. strojní vybavení podniku, flexibilnost výroby, finanční možnosti podniku atd., ale také vnějších faktorů, které ovlivňují efektivitu celého podniku jako propojeného systému. Tato situace může být ilustrována na následujícím příkladu výroby odlišných typů produktů, které se podnik ve smlouvě v jistém časovém horizontu zavázal dodat odběrateli (odběratelům). Pokud nebudou termíny dodávek splněny, podniku bude naúčtováno penále za každý výrobek, který nebyl dodán včas. Prvním problémem už je samotné rozhodování, zda podnik může přijmout zakázku vzhledem ke kapacitám stávající výroby. Pro hrubé odhady může sloužit jednoduché kapacitní plánování pomocí statických výpočtů. Tento propočet však ve většině případů nezahrnuje, anebo nelze do něj zahrnout procesy turbulentního prostředí ve výrobě, např. prostoj při náhlé poruše stroje, průtok úzkého místa ve výrobě a ostatní náhlé změny, které mohou ve výrobě nastat. K částečné predikci velikosti vlivu různých omezujících faktorů ve výrobě může být použita simulace. Pokud bychom měli jednoduše specifikovat pojem simulace, lze ho připodobnit k větě: „Co se stane, když ... ?“.

Cílem podniku je tedy nejlépe nakonfigurovat svůj systém (sestavit vhodný výrobní plán) tak, aby byly zakázky splněny. Při této konfiguraci musí být samozřejmě respektovány omezující faktory působící na podnik. Kvantifikaci úspěchu lze vyjádřit pomocí tzv. účelové funkce. Tato funkce bude odrážet kvalitu sestaveného výrobního plánu pomocí výše penalizace za jednotlivé nevyrobené produkty. Může se např. stát, že podnik nestihne vyrobit produkt v požadovaném časovém horizontu od odběratele, ale při jistém počtu nevyrobených produktů bude pro něj stále výhodné zakázku přijmout. Na druhou stranu, pokud by preferoval vyrábět tyto produkty co nejrychleji, vzrůstaly by podniku náklady na skladování díky předčasně vyrobeným produktům a byla by omezena stávající výroba.

Jak již bylo řečeno, tato problematika může být řešena pomocí simulace, tzn., bude sestaven simulační model virtuálně realizující výrobu odlišných typů produktů a na základě spočtené průběžné doby výroby produktu ze simulačního modelu (odezvy) by pomocí účelové funkce byla vyjádřena možná míra penalizace za nevyrobené produkty nebo za produkty, které by byly vyrobeny předčasně na sklad. Za pomoci lidského faktoru by byly u simulačního modelu měněny různé varianty výrobního plánu (vstupní parametry simulačního modelu) ve snaze minimalizovat výši penalizace. Obecně tuto problematiku zachycuje následující obrázek (viz Obr. 1-1).



Obr. 1-1 Klasický přístup k řešení optimalizačních problémů za pomoci lidského faktoru

Simulování všech možných kombinací vstupních parametrů simulačního modelu, za účelem určit optimální rozvrh výrobního plánu, je často nemožné nebo by byl zapotřebí neúměrně dlouhý výpočetní čas. Takové problémy zařazujeme do skupiny NP (nedeterministicky polynomiálních problémů), kdy náročnost řešení stoupá s každým novým vstupním parametrem simulačního modelu, tj. narůstá dimenze prostoru možných řešení.

Jedním z přístupů, jak řešit tento klasický problém, je částečná náhrada lidského faktoru pomocí **simulační optimalizace**, která vzniká propojením optimalizačních metod a simulačního modelu ve formě tzv. **simulačního optimalizátoru**. Simulační optimalizátor má za cíl určit vhodnou kombinaci vybraných hodnot vstupních proměnných simulačního modelu tak, aby bylo zajištěno co nejlepší uspořádání nebo chování simulačního modelu vzhledem ke zvolenému cíli. Simulační optimalizátor sice nezaručuje nalezení optimálního řešení, ale je schopen nalézt akceptovatelné řešení na základě definovaného cíle či více cílů.

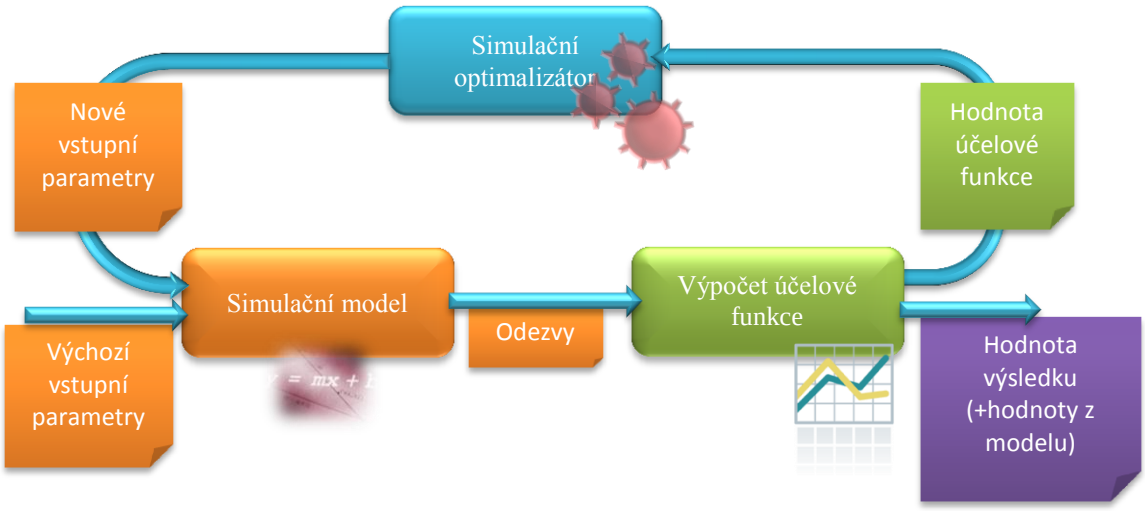
2 Simulační optimalizace

Jak již v úvodu bylo řečeno, simulační optimalizace (*Simulation Optimization*) je využívána zejména v případech, kdy tradiční rozhodování u problémů (zvláště u skupiny nedeterministicky polynomiálních problémů) pomocí lidského faktoru by bylo značně náročné nebo neproveditelné. Proto nejprve popíšeme oblast simulační optimalizace, na kterou je disertační práce zaměřena.

Optimalizaci prováděnou pomocí simulačního modelu lze rozdělit do dvou oblastí:

- **Modelová optimalizace** - vycházíme ze známého matematického popisu modelu, který prostřednictvím matematických vztahů nebo počítačových procedur určuje hodnoty účelové (někdy též kriteriální) funkce. Výpočty bývají většinou rychlé a proto je možné výpočet mnohokrát opakovat – lze si dovolit velký počet funkčních volání.
- **Technická optimalizace** - získáváme hodnoty účelové funkce, např. měřením, nebo **z výpočtů na simulačním modelu**. Doba potřebná pro zjištění hodnoty kriteriální funkce může být řádově sekundy, ale může dosáhnout i několika desítek minut nebo dnů (záleží to na složitosti konkrétního modelu nebo na použité počítačové technice). V praxi tato skutečnost vede k hledání sub-optimálních řešení s **relativně malým počtem opakování výpočtů**. Navíc při jednom výpočtu zjistíme jen jednu hodnotu účelové funkce bez poznání jejího okolí (o kterém by mnohé napověděla např. její derivace). [1]

Disertační práce je orientována především na technickou optimalizaci, kde simulační optimalizace je užívána tehdy, kdy by simulování všech možných kombinací vstupních parametrů simulačního modelu (za účelem určit optimální konfiguraci systému) bylo zhola nemožné nebo by byl zapotřebí neúměrně dlouhý výpočetní čas. Proto je jako částečná náhrada lidského faktoru používán tzv. simulační optimalizátor, který využívá optimalizačních metod globální optimalizace. Princip je názorně vysvětlen pomocí následujícího obrázku (viz Obr. 2-1). V následujících kapitolách budou podrobněji popsány jednotlivé prvky simulační optimalizace.



Obr. 2-1 Simulační optimalizace

V následující tabulce jsou uvedeny některé optimalizační softwary – simulační optimalizátory, které jsou v dnešní době k dispozici na trhu - viz Tabulka 2-1 ([2], [3], [4], [5], [6]).

Název optimalizačního balíku	Prodejce	Podpora simulačního softwaru	Použitá heuristická procedura
Issop	DUALIS	Arena eM-Plant/SIMPLE++ AUTOMOD DOSIMIS Quest	**Genetické algoritmy**
Evolutionary Optimizer	Imagine That	Extend	**Genetické algoritmy**
Evolver	Palisade	@Risk	**Genetické algoritmy, Simulované žíhání**
OptQuest	OptTek Systems	AnyLogic Arena Crystal Ball CSIM19 Enterprise Dynamics Micro Saint ProModel Quest SimFlex SIMPROCESS SIMUL8 TERAS	**Rozptylové prohledávání (Scatter Search), Zakázané prohledávání (Tabu Search), Lineární/Celočíselné programování (Linear/Integer Programming), Neuronové sítě (Neural Networks)**
WITNESS Optimizer	**Lanner Group**	**WITNESS**	**Horolezecký algoritmus (Hill Climbing), Adaptivní simulované žíhání (Adaptive Thermostatistical Simulated Annealing), Zakázané prohledávání, Min/Mid/Max-tests, Algoritmy Six Sigma,**

Tabulka 2-1 Ukázka optimalizačního software na trhu

A nyní ke struktuře disertační práce. Ve dvou následujících kapitolách je popsán současný stav problematiky simulační optimalizace. Podrobně jsou popsány zejména optimalizační algoritmy, které se používají pro řešení úloh globální optimalizace při řízení simulačních experimentů zaměřených na diskrétní výrobní systémy a výrobní procesy. Tyto známé algoritmy (po eventuálních drobných modifikacích) jsou využity v dalších kapitolách, které tvoří jádro vlastní disertační práce. Na závěr disertační práce jsou uvedeny přínosy řešení a nástin dalšího postupu prací.

3 Současný stav problematiky

Simulační model je abstraktní reprezentace systému, obvykle obsahující strukturní, logické anebo matematické vztahy, které popisují systém pomocí stavu, entit a jejich atributů, množin, procesů, událostí, aktivit a zpoždění. [7]

Jinými slovy lze také říci, že simulační model je funkcí (jejíž explicitní forma nemusí být známa), která vyhodnocuje soubory technických dat, které jsou reprezentovány pomocí souboru hodnot. Simulační model může být také definován na základě předpisu funkce. V případě simulační optimalizace jsou pro simulační model zpravidla definovány tři základní elementy:

1. **Rozhodovací proměnné** – vstupní parametry simulačního modelu. Hodnoty parametrů se pohybují v rámci definovaných mezí.
2. **Účelová funkce** – vyjádření cíle simulačních experimentů, ohodnocení výsledků z modelu.
3. Specifikovaná **omezení** – omezení vztahující se na rozhodovací proměnné nebo účelovou funkci.

Ve světě optimalizace je někdy místo termínu **rozhodovací proměnné** užíván termín řídící proměnné – *Controls* nebo faktory – *Factors.* Synonymem **odezev** (*Responses*) bývá termín výstupy - *Outputs.*

3.1 Vstupy simulačního modelu - rozhodovací proměnné

Jak již bylo řečeno, simulační optimalizátor se snaží pomocí zvolené optimalizační metody (implementované ve formě algoritmu) nalézt nejlepší variantu, tj. kombinaci vybraných hodnot vstupních parametrů – hodnoty **rozhodovacích proměnných** (*Decision Variables*) - simulačního modelu tak, aby bylo zajištěno nejlepší uspořádání nebo chování modelu vzhledem ke zvolenému kritériu (viz Obr. 2-1).

Poznámka: Za účelem sladění pojmů používaných v simulační optimalizaci, bude v dalším textu namísto „nastavení vstupních parametrů simulačního modelu“ použito „nastavení hodnot rozhodovacích proměnných“.

Definujme nejprve **prostor všech řešení** X (*Problem Space*), který je definičním oborem účelové funkce. Hodnota účelové funkce (v případě vícekriteriální optimalizace hodnoty více účelových funkcí) vyjadřuje, nakolik jsme se přiblížili k vytýčenému cíli.

Dále definujme podprostor – **prohledávaný prostor** $\tilde{X}$ (*Search Space*), ve kterém se bude pohybovat optimalizační metoda za účelem nalezení optimálního řešení ve formě globálního extrému účelové funkce (tento pojem bude vysvětlen v dalších kapitolách). Prohledávaný prostor je tedy prohledávaná (testovaná) část prostoru všech řešení X (je vlastní podmnožinou množiny všech řešení tj. $\tilde{X} \subseteq X$), jejíž velikost může být definována např. ve formě hranic pro každou osu - n-rozměrného kvádru (*Box Constraint*):

$$\tilde{X} = \prod_{j=1}^{n} \tilde{X}_j = \prod_{j=1}^{n} [a_j, b_j],\ a_j \leq b_j \tag{3.1}$$

kde:

- $\tilde{X}$ … Prohledávaný prostor optimalizační metodou.
- $\prod_{j=1}^{n} [a_j, b_j]$ … kartézský součin konečného počtu n množin (intervalů v hranaté závorce).
- $\tilde{X}_j$ … Prohledávaný interval j-té rozhodovací proměnné.
- j… Index j-té rozhodovací proměnné.
- n… Dimenze prostoru všech rozhodovacích proměnných.
- a_j … Dolní mez prohledávaného j-tého intervalu.
- b_j … Horní mez prohledávaného j-tého intervalu.

Optimální řešení představuje takové nastavení vstupních parametrů simulačního modelu – hodnot **rozhodovacích proměnných**, aby bylo dosaženo nejlepšího chování simulačního modelu, vzhledem ke specifikovanému **cíli** (kritériu), který je vyjádřen pomocí **účelové funkce** a zároveň toto řešení musí splňovat podmínku **přípustnosti řešení** – musí vyhovovat definovaným **omezením**.

Rozhodovacími proměnnými simulačního modelu mohou být např. počet strojů, počet lidí, počet entit atd. Rozhodovací proměnné jsou tedy takové faktory, které mají v simulačním modelu vliv na výsledný výstup z tohoto modelu - **odezvu**, která je následně argumentem účelové funkce.

Hodnoty rozhodovacích proměnných jsou měněny po proběhnutí simulačního experimentu (jedna iterace). Tyto hodnoty také musí splňovat definovaná omezení, např. být v rozsahu své dolní a horní meze atd. V simulační optimalizaci jsou jejich hodnoty nejčastěji nastavovány pomocí optimalizačních algoritmů využívajících převážně heuristických metod globální optimalizace. Hodnoty rozhodovacích proměnných jsou měněny do té doby, dokud:

- Nejsou splněna jejich definovaná omezení – pokud navrhované řešení nevyhovuje některému z definovaných omezení, může být vygenerováno jiné řešení nebo může být použit opravný algoritmus.
- Není splněno kritérium ukončení (zastavení optimalizačního experimentu, pokud je splněna některá z podmínek, které bývají většinou konjunktivní např. z důvodu, že se nedaří nalézt lepší hodnotu účelové funkce, než dosud známá nejlepší nalezená hodnota účelové funkce).

Rozhodovací proměnné nabývají v odlišných variantách systému různých hodnot, tzv. úrovní. Rozhodovací proměnné mohou být:

- Kvalitativní - např. řád fronty, typ rozdělení, různá pravidla pro pohyb entit aj. V určitém případě může jít o srovnání variant se zcela odlišnou strukturou. Pak je jedinou určující rozhodovací proměnnou typ simulovaného systému.
- Kvantitativní - numerické hodnoty. Lze je dále dělit z hlediska typu:
    - *Diskrétní* (počet obslužných zařízení, kapacita fronty) – u diskrétní proměnné můžeme předpokládat celočíselné nebo reálné hodnoty. U reálných čísel musí být definována velikost kroku. Odchylka po sobě jdoucích hodnot diskrétních rozhodovacích proměnných se pohybuje v daném rozsahu jednotlivých os. Např. diskrétní rozhodovací proměnná s rozsahem [0,2] s krokem 0.25 bere v úvahu hodnoty 0; 0.25; 0.5; 0.75; 1.0; 1.25; 1.5; 1.75 a hodnotu 2.0. [8]
    - *Spojité* (doba obsluhy, doba bezporuchového provozu), které mohou obvykle nabývat jakýchkoliv hodnot z množiny nezáporných reálných čísel.

Poznámka: Alternativní značení intervalu[1] - místo špičatých závorek $\langle a, b\rangle$ používané v české literatuře se v cizojazyčné literatuře pro uzavřené intervaly používají hranaté závorky $[a, b]$. Aby byla zachována autentičnost citace, některé vzorce budou obsahovat toto alternativní značení. Protože v dalších kapitolách budou využívány množiny, u kterých jsou jednotlivé prvky odděleny čárkou, bude u reálných čísel namísto desetinné čárky používána desetinná tečka.

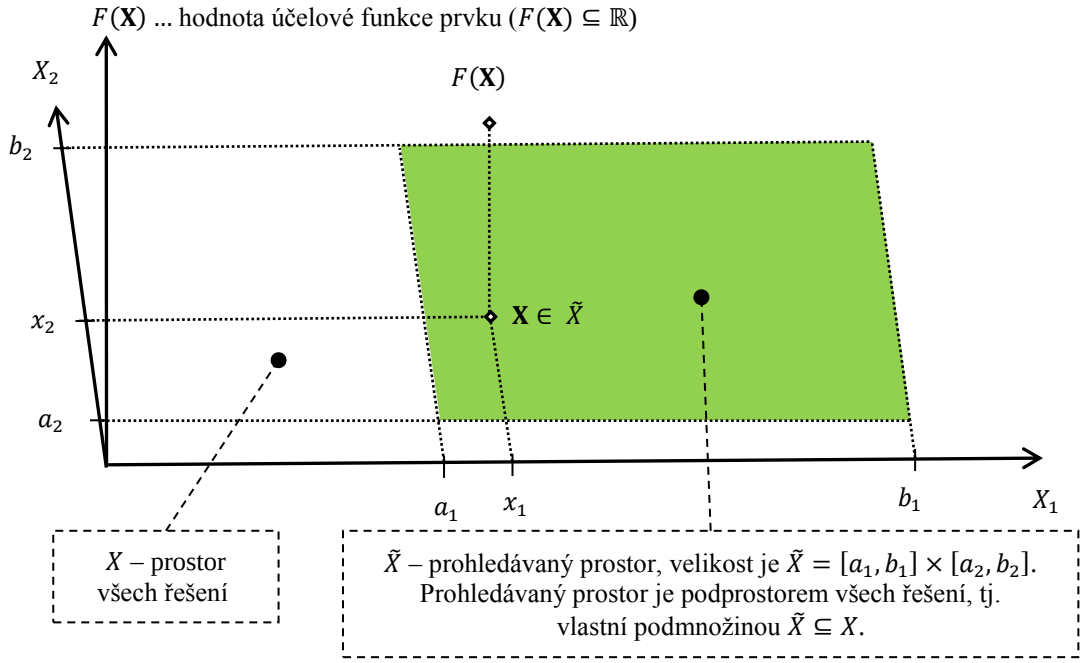
Hodnoty jednotlivých rozhodovacích proměnných jsou uloženy ve vektoru $\mathbf{X}$. Tento vektor lze také obecně nazvat prvkem $\mathbf{X}$ v prohledávané části prostoru $\tilde{X}$. Prvek je tedy n-dimensionální vektor $\mathbf{X}$ a graficky vyjadřuje bod v prohledávaném prostoru o souřadnicích:

$$\mathbf{X} = [x_1, x_2, \ \dots \ , x_n] \tag{3.2}$$

[1] http://cs.wikipedia.org/wiki/Interval_(matematika)

kde:

- **X** … Prvek (vektor) obsahující hodnoty jednotlivých rozhodovacích proměnných.
- x_1 … Hodnota první rozhodovací proměnné – v grafu účelové funkce představuje hodnotu souřadnice na 1. ose - první rozhodovací proměnné, tj. X_1, v prohledávaném prostoru.
- n… Dimenze prostoru všech rozhodovacích proměnných.



Obr. 3-1 Dvourozměrný prohledávaný prostor

Je logické, že čím bude vyšší počet rozhodovacích proměnných, tím bude vzrůstat časová náročnost výpočtu optimalizačního algoritmu.

Poněvadž v disertační práci u některých optimalizačních algoritmů (popsaných ve formě pseudopascalu) a také u matematických zápisů bude zapotřebí přistupovat k jednotlivým složkám prvku (hodnotám souřadnic rozhodovacích proměnných), bude prvek transformován na tzv. **seznam**, kde hodnoty souřadnic rozhodovacích proměnných prvku budou indexovány podle pořadí os jednotlivých rozhodovacích proměnných od indexu 0 až do indexu počet os – 1:

$$\mathbf{X}[j] = x_j \forall j: j = \{0,1,2, \dots, n-1\} \tag{3.3}$$

kde:

- x_j … Hodnota souřadnice bodu – hodnota j-té rozhodovací proměnné prvku **X**.
- : … Popis vlastnosti, tj. …. kde (který) …. .
- j… Index j-té rozhodovací proměnné – index osy v grafu prohledávaného prostoru.
- n… Dimenze prostoru všech rozhodovacích proměnných.

Předchozí zápis upravuje zápis rovnice (3.2), kde namísto indexu 1 u hodnoty první rozhodovací proměnné je použit index 0 a tudíž poslední index je roven $n-1$. Takový přístup je zaveden z důvodu sladění principů užívaných v programování, kde indexy prvků seznamu jsou číslovány od nuly. Jednotlivé metody pro práci se seznamy budou detailně popsány v kapitole práce se seznamy (viz kapitola 3.4 Seznam).

Poznámka: V obrázcích grafu prohledávaného prostoru, popřípadě grafu účelové funkce, bude zachováno indexování os a prvků od hodnoty 1, protože by bylo nezvyklé značit např. první osu v prohledávaném prostoru indexem 0. Indexování od nuly bude využito zejména v matematických zápisech nebo v algoritmech.

U většiny metod (zejména v případě evolučních metod) je zpravidla generováno více prvků, namísto jediného prvku. Tyto prvky jsou následně nějakým způsobem dále zpracovány. Abychom tyto prvky navzájem od sebe odlišily, je u prvků použita indexace. Pro i-tý prvek a jeho souřadnice (pro n-dimensionální prostor rozhodovacích proměnných) tedy platí:

$$\mathbf{X}_i = \left[{}_{(i)}x_j\right] \forall {}_{(i)}x_j \in \mathbf{X}_i \wedge \forall {}_{(i)}x_j \in \tilde{X}_j : i = \{0,1,2,\dots,m-1\}, j = \{0,1,2,\dots,n-1\} \tag{3.4}$$

kde:

- $\mathbf{X}_i$ … i-tý prvek.
- i… Index prvku.
- ${}_{(i)}x_j$ … Hodnota j-té rozhodovací proměnné X_j. Geometricky ${}_{(i)}x_j$ znázorňuje souřadnici i-tého prvku na ose X_j v prohledávaném prostoru. Jedná se o složku prvku - vektoru $\mathbf{X}_i$ (generovaného v i-té iteraci) v n-rozměrném prohledávaném prostoru $\tilde{X}$.
- $\wedge$ … Logický součin, konjunkce, logická spojka „a“ zároveň
- $\tilde{X}_j$ …Prohledávaný interval j-té rozhodovací proměnné – hrana prohledávaného prostoru $\tilde{X}$.
- j… Index j-té rozhodovací proměnné – index osy.
- n… Dimenze prostoru všech rozhodovacích proměnných – v rámci této disertační práce bude dimenze prostoru všech rozhodovacích proměnných rovna dimenzi prohledávaného prostoru.
- m … Počet vygenerovaných bodů v jednom kole algoritmu – počet iterací (v kontextu evolučních algoritmů - počet vygenerovaných jedinců v populaci).

Protože u algoritmů se zpravidla generuje více prvků (pomocí cyklů), tyto prvky budou postupně umístěny do seznamu, podle pořadí ve kterém byly vygenerovány (seznam kontextu evolučních algoritmů představuje populaci). Index prvku tedy určuje pořadí prvku v seznamu:

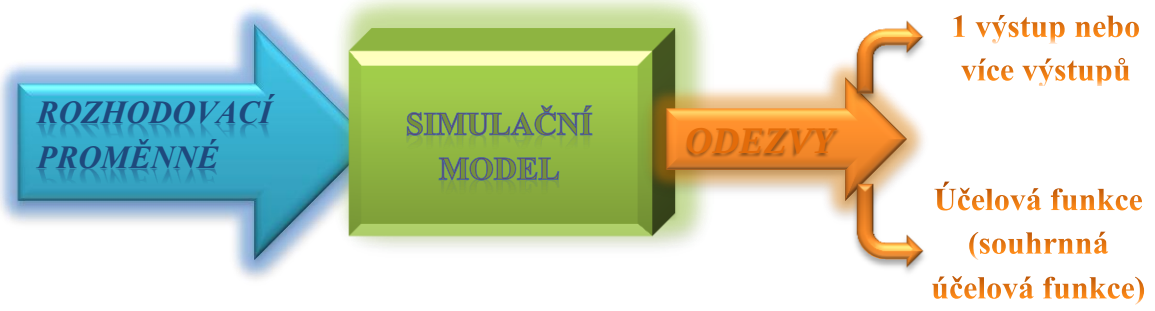
$$\mathbf{X}_i = S[i] \forall i : i = \{0,1,2,\dots,m-1\} \tag{3.5}$$

kde:

- S … Seznam prvků.
- m … Velikost (délka) seznamu S. U algoritmů bude délka seznamu získána pomocí funkce Length tj. $m = \mathrm{Length}(S)$.
- i … Index (pozice) prvku v seznamu.

3.2 Výstupy simulačního modelu - odezvy, účelová funkce

Díky výstupům simulačního modelu – **odezev** - získaných po provedení simulačního běhu, lze získat hodnoty pro specifikovanou účelovou funkci. V simulační optimalizaci se často snažíme **maximalizovat** nebo **minimalizovat** jednotlivé odezvy, jednotlivou účelovou funkci nebo souhrnnou účelovou funkci, která sdružuje více účelových funkcí v případě vícekriteriální optimalizace.



Obr. 3-2 Vyhodnocení výstupů ze simulačního modelu

Jednotlivé odezvy jsou funkcí prvku $\mathbf{X}$ reprezentujícího nastavené hodnoty u vstupních parametrů simulačního modelu, tedy hodnoty jednotlivých rozhodovacích proměnných.

$$O = \{o_0(\mathbf{X}), o_1(\mathbf{X}), \dots, o_{q_o-1}(\mathbf{X})\} \tag{3.6}$$

kde:

- O … Množina odezev.
- $o_0(\mathbf{X})$ … První odezva ze simulačního modelu.
- $\mathbf{X}$ … Prvek (vektor) obsahující hodnoty jednotlivých rozhodovacích proměnných.
- q_o … Počet odezev simulačního modelu.
- $o_{q_o-1}(\mathbf{X})$… Poslední odezva simulačního modelu.

Na základě odezev ze simulačního modelu lze popsat hodnotu účelové funkce jako:

$$y = F(O(\mathbf{X})) \tag{3.7}$$

kde:

- y … Hodnota účelové funkce.
- F … Účelová funkce.
- O … Množina odezev.
- $\mathbf{X}$ … Prvek obsahující hodnoty jednotlivých rozhodovacích proměnných.

Z předchozího zápisu je patrné, že hodnota účelové funkce je závislá na nastavených hodnotách vstupních parametrů simulačního modelu. Po vyhodnocení simulačního experimentu je získána odezva/y, která je argumentem pro výpočet hodnoty účelové funkce. Optimalizační metody se tedy snaží generovat jednotlivé prvky v prohledávaném prostoru, které budou ohodnoceny hodnotou účelové funkce, která je získána na základě odezvy simulačního modelu. Optimalizační metoda postupně mapuje průběh účelové funkce za účelem maximalizovat nebo minimalizovat hodnoty účelové funkce. Během mapování terénu účelové funkce musí optimalizační metoda respektovat všechna nadefinovaná omezení (viz Obr. 3-3 Příklad grafu účelové funkce).

Poznámka: V dalším popisu budeme využívat jedinou účelovou funkci, která bude značena $F(\mathbf{X})$.

Na **monokriteriální** optimalizaci (*Single-Objective Optimization* - optimalizaci jedné účelové funkce $F(\mathbf{X})$) lze pohlížet jako na geometrický problém ($n+1$-rozměrný prostor, kde $n+1$ dimenze slouží jako návratová hodnota účelové funkce), ve kterém se hledá řešení na n-rozměrné ploše ve formě minima nebo maxima a kdy

toto řešení musí splňovat všechny z nadefinovaných podmínek omezení (viz Obr. 3-3 Příklad grafu účelové funkce).

Většinou ale bývá situace mnohem složitější, zejména při problémech praktického charakteru. Účelová funkce může obsahovat více stejných globálních extrémů nebo v náročnějším případě argumenty účelové funkce jsou spojité hodnoty:

$$F: \tilde{X} \mapsto \mathbb{R}, \tilde{X} \subseteq \mathbb{R} \tag{3.8}$$

V případě, že existence více jak jednoho globálního extrému není únosná, je vhodné problém přeformulovat. Na druhou stranu však více globálních extrémů skýtá možnost nalezení více stejně kvalitních řešení, z nichž si řešitel může a posteriori vybrat. [9]

Mnohdy je při optimalizaci i více účelových funkcí, které mohou být navzájem protichůdné např. minimalizace průběžné doby výroby vs. maximalizace zisku. V tomto případě se jedná o víceúčelovou optimalizaci (*Multi-objective Optimization)*, též nazývanou vícekriteriální hodnocení variant. Jedná se tedy o úlohu hledání optima pro **množinu jednotlivých účelových funkcí** vyjadřující jednotlivá kritéria:

$$F_l \in \zeta, l = \{0,1,2, \dots, q-1\} \tag{3.9}$$

kde:

- F_l … l-tá účelová funkce.
- l … Index účelové funkce.
- F_0 … První účelová funkce.
- q … Počet účelových funkcí.

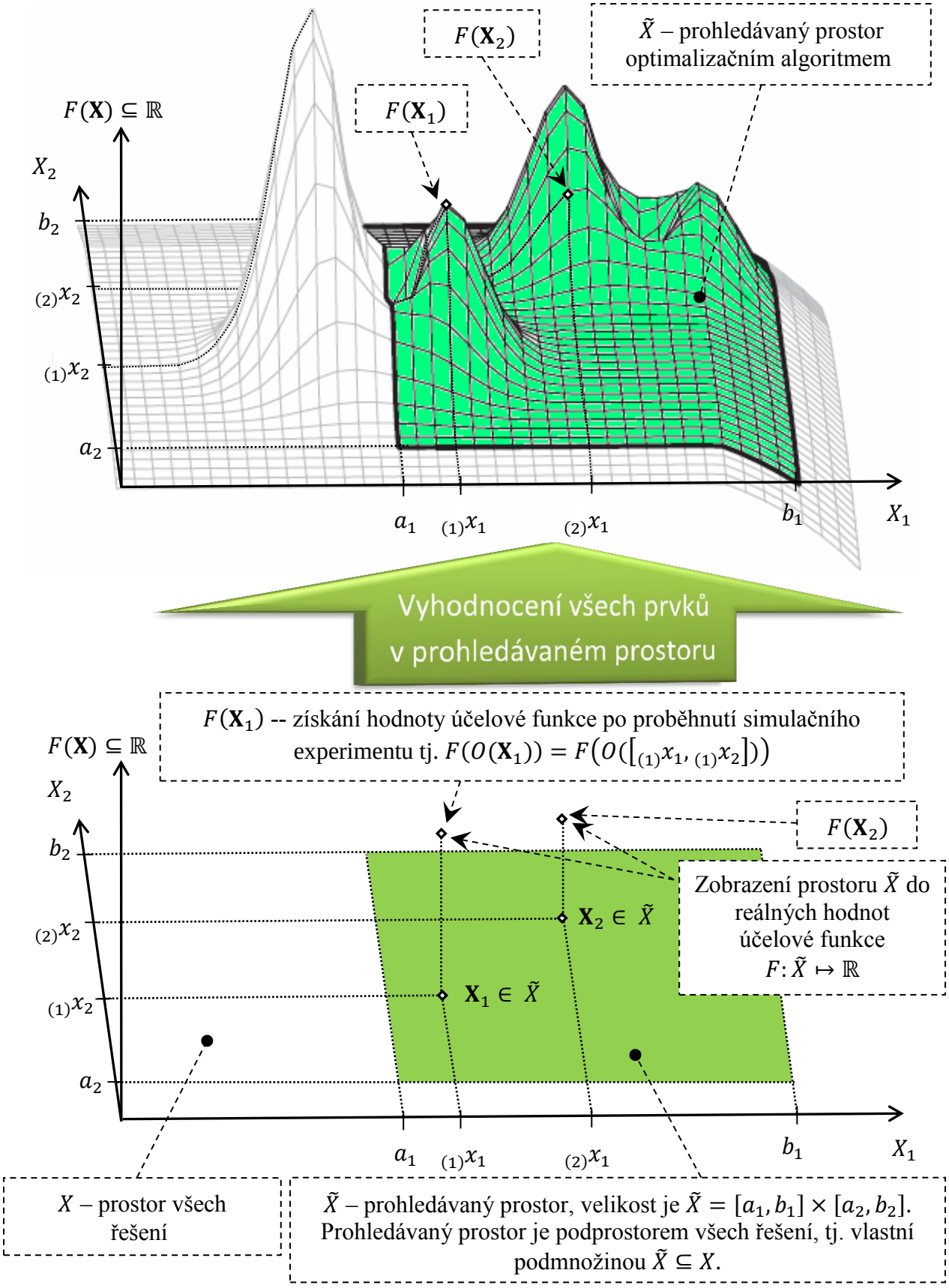
Hodnoty jednotlivých účelových funkcí pro jednotlivé prvky vygenerované optimalizačním algoritmem, jejichž hodnoty jsou získány ze simulačního experimentu, lze agregovat do vektoru hodnot účelových funkcí:

$$\mathbf{Y}_i = \left[y_{i,0}, y_{i,1}, \dots, y_{i,q-1}\right], \mathbf{Y}_i \subseteq \mathbb{R}, i = \{0,1,2, \dots, m-1\} \tag{3.10}$$

kde:

- $\mathbf{Y}_i$ … Vektor hodnot účelových funkcí.
- i … Index vektoru hodnot účelových funkcí.
- $y_{i,0}$ … Hodnota první účelové funkce v i –tém vektoru $\mathbf{Y}_i$.
- $\mathbb{R}$ … Množina - obor reálných čísel.
- q … Počet účelových funkcí.
- m … Počet vygenerovaných prvků v jednom kole algoritmu.

Pro názornost uveďme jednoduchou metodu výpočtu **váženého součtu** (*Weighted Sum)* užívanou v oblasti víceúčelové optimalizace. **Váhy** vyjadřují důležitost jednotlivých účelových funkcí a také určují, zdali funkce má být maximalizována ($w_l > 0$) nebo minimalizována ($w_l < 0$). Užitím této transformace je problém víceúčelové optimalizace redukován na optimalizaci **jednoho** účelu (cíle) - monokriteriální rozhodovací úloha.



Obr. 3-3 Příklad grafu účelové funkce – zobrazení

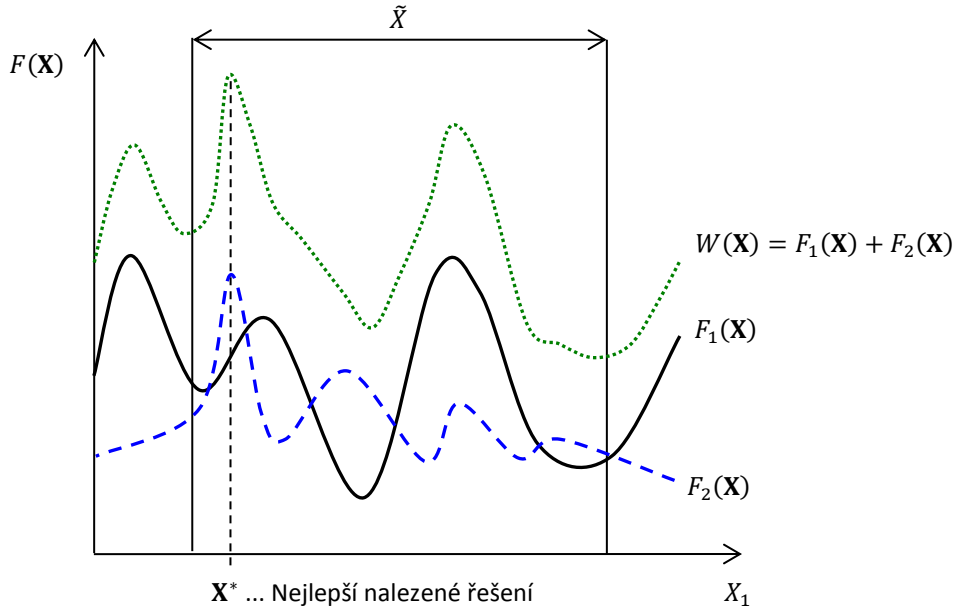
Namísto několika hodnot účelových funkcí pro konkrétní řešení je vypočtena jedna hodnota vyjadřující atraktivitu daného řešení pro všechny účelové funkce.

$$W(\mathbf{X}) = \sum_{l=1}^{q} w_l F_l(\mathbf{X}) \tag{3.11}$$

kde:

- $W(\mathbf{X})$ … Hodnota váženého součtu pro prvek $\mathbf{X}$.
- q … Počet účelových funkcí.
- w_l … Váha l-té účelové funkce.
- l … Index účelové funkce.
- $F_l(\mathbf{X})$ … Hodnota l-te účelová funkce pro prvek $\mathbf{X}$.

Simulační experimenty jsou prováděny nad prohledávaným prostorem $\tilde{X}$, (kde $\mathbf{X} \in \tilde{X}, \tilde{X} \subseteq X$ za respektování definovaných dalších omezení – viz Obr. 3-4 Příklad metody váženého součtu). Váhy jsou v tomto příkladu nastaveny na hodnotu 1 a jedná se o maximalizaci obou funkcí $F_1(\mathbf{X})$ a $F_2(\mathbf{X})$. Z obrázku je patrné nalezené jediné optimum $\mathbf{X}^*$. Nad tímto prostorem je aplikována metoda váženého součtu uplatněná na obě účelové funkce, která poskytuje pro každý prvek hodnotu váženého součtu, kterou popisuje výslednou hodnotu $W(\mathbf{X})$.



Obr. 3-4 Příklad metody váženého součtu

Prvky v prohledávaném prostoru $\tilde{X}$ funkce $F(\mathbf{X})$ mohou být reprezentovány jako čísla, seznamy apod., v závislosti na optimalizačním problému. Účelové funkce nemusí být nezbytně pouhými matematickými výrazy, ale mohou být algoritmy, které mohou např. zahrnovat numerickou simulaci.

Pokud by byl svět jednoduchý a budoucnost by se dala předvídat, všechna data v optimalizačním modelu by byla konstantní a model by byl deterministický. Je ale zřejmé, že deterministický optimalizační model nemůže zachytit všechny relevantní komplikovanosti a spletitosti v oblasti praktického rozhodování. V modelu proto používáme pravděpodobností rozdělení, díky nimž může být zachycena reálnost modelovaného systému. Pokud jsou data v modelu náhodná (bývají popsána pouze s jistou pravděpodobností), hodnota účelové funkce

také bude mít jen jistou pravděpodobnost pro kterékoliv zvolené hodnoty rozhodovacích proměnných. Znamená to tedy, že i odezvy jsou náhodné proměnné.

Je také nutné si uvědomit, že obdržené výsledky simulačního běhu jsou většinou statistiky - souhrnné hodnoty náhodných rozdělení (rozložení četnosti) výsledků modelu, které jsou vyjádřeny například jako střední hodnota, rozptyl nebo směrodatná odchylka.

3.3 Omezení

Protože i my (a také počítače) jsme při výpočtech limitováni pamětí a časem, nelze využívat optimalizační metody, které prohledávají celý rozlehlý prostor všech řešení X, potažmo jeho podmnožinu - prohledávaný prostor $\tilde{X}$. Jsou proto definována různá **omezení** (*Constraints*), abychom tento prostor zmenšili. Dalším důvodem pro definování omezení je, že omezení více reflektují reálnou modelovanou situaci, podle níž je sestaven model. Respektování omezení, pokud jsou definována, je nutností, která musí být vždy splněna. Zvyšováním počtu takových omezujících podmínek se zvyšuje reálnost výsledků získaných pomocí simulačního modelu.

Pokud řešení vyhovuje nadefinovaným podmínkám, jedná se o tzv. **přípustné** (možné, proveditelné) **řešení**. Neproveditelnost nastává v případě, že žádnou kombinací hodnot rozhodovacích proměnných nelze vyhovět souboru omezení. Je také zapotřebí dostatečně analyzovat model a řešený problém, zda je vůbec možné dosáhnout přípustného řešení. [8]

Pokud není úloha omezena žádnými podmínkami, hledá se **extrém volný** (*Free Optimization Problem*). Při existenci omezujících podmínek hledáme **extrém vázaný** (*Constraint Optimization Problem*).

Daný prohledávaný prostor $\tilde{X}$, který je prohledáván optimalizačním algoritmem, je často omezen díky definovaným omezením na vlastní podmnožinu (podprostor) přípustných řešení:

$$X_{Feasible} \subseteq \tilde{X} \subseteq X \tag{3.12}$$

kde:

- $X_{Feasible}$ … Prostor přípustných řešení.
- $\subseteq$ … Vlastní podmnožina (v případě $X_{Feasible} \subseteq \tilde{X}$ množina $X_{Feasible}$ obsahuje stejné prvky jako nadmnožina $\tilde{X}$, ale není rovna množině $\tilde{X}$).
- X … Prostor všech řešení.

Pro množinu **nepřípustných řešení** tedy platí:

$$X_{Non-feasible} = X \setminus X_{Feasible} \tag{3.13}$$

kde:

- $\setminus$ … Rozdíl dvou množin označuje takovou množinu obsahující každý prvek, který se nachází v první z množin, ale nenachází se ve druhé množině.[2]

Nepříjemný případ nastává tehdy, jestliže uplatněním omezujících podmínek vznikají jednotlivé ostrůvky disjunktních podmnožin v prostoru X.

Omezení popisují vztahy mezi rozhodovacími proměnnými a omezují hodnoty těchto proměnných a také účelové funkce. Taková omezení jsou:

[2] http://cs.wikipedia.org/wiki/Rozd%C3%ADl_mno%C5%BEin

- Omezení vztažená na **rozhodovací proměnné**:
  - **Logická** - např. součet ploch strojů (počet strojů byl určen optimalizačním algoritmem) nesmí být větší než plocha dílny.
  - **Praktická** – např. vyhodnocování „všech“ argumentů účelové funkce, které jsou podmnožinou oboru reálných čísel $\mathbf{X} \in \mathbb{R}$ (např. hranice prohledávaného prostoru).
- Omezení vztažená na **účelovou funkci** - omezení vyplývají ze situace, která je řešena pomocí modelu.

3.3.1 Omezení vztažená na rozhodovací proměnné

Je zřejmé, že díky definování různých **omezujících podmínek** se prohledávaný prostor zmenší. Ve většině reálných případů se tento prostor při definované omezující podmínce rozdělí na **množinu přípustných řešení** vyhovující podmínce a na **množinu nepřípustných řešení** nevyhovující podmínce. Obecný zápis množiny přípustných řešení splňující definované omezení lze formulovat takto:

$$X_{Feasible}(g(\mathbf{X})) = \{\mathbf{X} \in X : g(\mathbf{X}) \geq 0\} \tag{3.14}$$

kde:

- $g(\mathbf{X})$ … Funkce omezení kladené na rozhodovací proměnnou.

Množinu přípustných řešení $X_{Feasible}$ můžeme definovat jako řešení, které vyhovuje všem specifikovaným (většinou konjunktivním) podmínkám. Ve většině případů je definována více než jedna omezující podmínka.

Omezení kladené na rozhodovací proměnnou má většinou některý z následujících tvarů:

$$g(\mathbf{X}) \leq q_g \tag{3.15}$$

$$g(\mathbf{X}) = q_g \tag{3.16}$$

$$g(\mathbf{X}) \geq q_g \tag{3.17}$$

kde:

- q_g … Známá konstanta.

Pro lepší představu je někdy lepší převést konstantu na levou stranu a zahrnout ji do funkce omezení $g(\mathbf{X})$ aby na pravé straně zůstala nula. [9]

3.3.2 Omezení vztažená na účelovou funkci

Příkladem omezení kladeného na účelovou funkci může být respektování smluvních podmínek týkajících se odbytu svých produktů vůči zákazníkům. Podnik se snaží vyrobit co nejvíce produktů, ale v takovém maximálním množství, které je zákazník ochoten odebrat. Účelová funkce popisuje množství vyrobených produktů. Účelová funkce je závislá na výstupu simulačního modelu. Podnik se snaží maximalizovat objem výroby a ten je závislý na počtu dělníků a počtu strojů:

$$X_1 \in \mathbb{N}, X_2 \in \mathbb{N} \tag{3.18}$$

kde:

- X_1 … Počet dělníků.
- X_2 … Počet dělníků.
- $\mathbb{N}$ … Přirozená čísla.

Tyto rozhodovací proměnné v grafu účelové představují osy X_1 a X_2. U simulačního modelu, který zachycuje výrobu produktů, budou pomocí optimalizační metody měněny rozhodovací proměnné (vstupní parametry simulačního modelu), které budou ovlivňovat výši produkce podniku. Obecně lze omezení kladené na účelovou funkci vyjádřit:

$$g\big(F(\mathbf{X})\big) \leq q_g \tag{3.19}$$

$$g\big(F(\mathbf{X})\big) = q_g \tag{3.20}$$

$$g\big(F(\mathbf{X})\big) \geq q_g \tag{3.21}$$

kde:

- q_g … Známá konstanta.

Účelovou funkci, v případě předchozí úlohy, lze vyjádřit pomocí následující nerovnice:

$$g\big(F(\mathbf{X})\big): F(\mathbf{X}) > Q \tag{3.22}$$

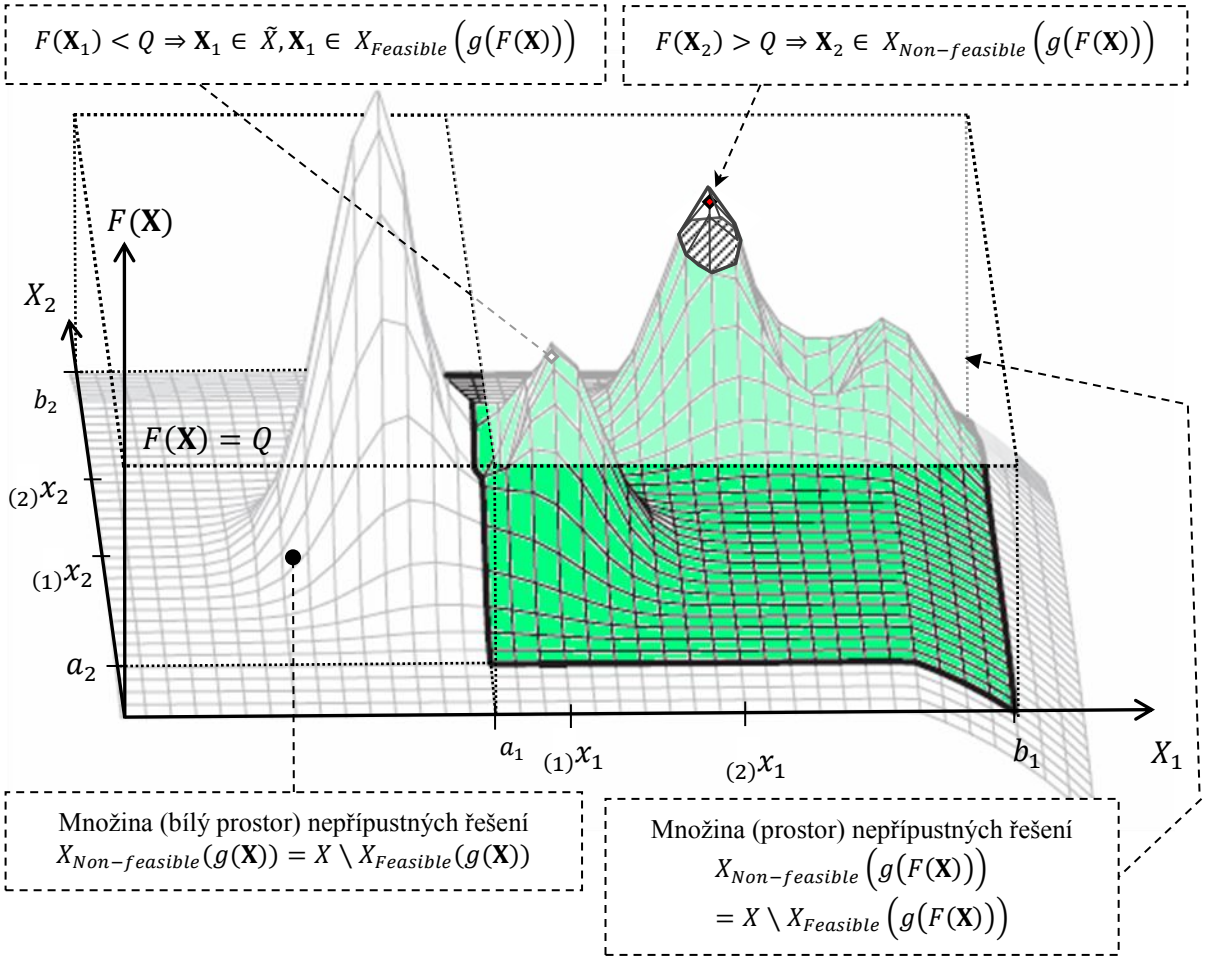
kde:

- $F(\mathbf{X})$ … Účelová funkce, kterou se snažíme maximalizovat – výše produkce.
- Q … Množství produktů odebrané odběratelem.

Omezení kladené na rozhodovací proměnné je takové, že podnik může uvolnit ze stávající produkce společnosti na operace související s výrobou produktu maximálně určitý počet lidí. To samé platí pro počet strojů, který se může pohybovat jen v určitém intervalu. Prvek tedy musí být prvkem prohledávaného prostoru, viz předchozí příklad. Pokud bychom chtěli vyjádřit předchozí zápis omezení kladeného na účelovou funkci pomocí rozhodovací podmínky, lze zapsat:

$$g\big(F(\mathbf{X})\big) \equiv \begin{cases} \mathbf{X} \in X_{Non-feasible}\left(g\big(F(\mathbf{X})\big)\right) & \text{jestli } F(\mathbf{X}) > Q \\ \mathbf{X} \in X_{Feasible}\big(g\big(F(\mathbf{X})\big)\big) & \text{jinak} \end{cases} \tag{3.23}$$

Na dalším obrázku (viz Obr. 3-5) je v grafu účelové funkce zachycena podstata nastíněného problému. Je patrné, že prvek $\mathbf{X}_2$ nesplňuje definované omezení kladené na účelovou funkci a tudíž spadá do oblasti nepřípustných řešení. Hodnota účelové funkce vyjadřující množství vyrobených produktů tohoto prvku totiž přesahuje stanovené množství odebraných výrobků ve smlouvě.



Obr. 3-5 Příklad grafu účelové funkce s omezením účelové funkce

3.3.3 Metody práce s omezujícími podmínkami

V některých mezních případech může dojít k situaci, že všechny vygenerované prvky, v rámci iterace optimalizačního algoritmu, budou nepřípustné. Jedná se zejména o takové prvky, které nevyhoví omezením kladeným na účelovou funkci (to jestli spadá prvek do množiny přípustných řešení, se dozvíme až po vypočtení simulačního experimentu). Pokud bychom taková řešení zamítli, optimalizační algoritmus nebude mít žádnou informaci o tom, jakým směrem se má vydat na své cestě za hledáním optimálního prvku. Proto je možné využít metody práce s různými typy omezení, které jsou zpravidla používány v kontextu s evolučními výpočty (zpravidla s vícekriteriální optimalizací). Tyto metody lze samozřejmě použít i v monokriteriální optimalizaci. Takovými metodami jsou např.:

1. **Trest smrti** (*Death Penalty*)
2. **Penalizační funkce** (*Penalty Function*)
3. **Opravné algoritmy** (*Repair Algorithms*) **a speciální operátory**
4. **Dekodéry** (*Decoders*)

3.3.3.1 Trest smrti (Death Penalty)

Trest smrti znamená přímé odmítnutí všech prvků z oblasti nepřípustných řešení. Díky zamítnutí se prvek nezahrnuje do dalších optimalizačních procesů. Toto omezení má opodstatnění v oblastech nepřípustných řešení, které jsou rozlehlé. Stinnou stránkou zavržení nepřípustných prvků je však ztráta informací, které mohly být získány a nebudou tak použity během optimalizace. [10]

V případě, kdy vygenerovaný prvek pomocí optimalizačního algoritmu leží mimo meze prohledávaného prostoru $\tilde{X}$, následuje generování nového prvku do té doby, dokud nebude ležet uvnitř prohledávaného prostoru. Překročení hranic prohledávaného prostoru je často způsobeno tím, že nové prvky (v grafu účelové funkce body), které jsou zkoumány, jsou generovány za pomoci veličin náhodného rozdělení.

Následující algoritmus (viz Algoritmus 3-1) popisuje jak generovat nový prvek na základě rovnoměrného rozdělení.

Poznámka: Text v odstavci uvedený za znakem ❖ popisuje metody (funkce, procedury). V jednotlivých algoritmech za znakem // bude uváděn jednořádkový komentář. Pokud se komentář nevejde na jednu řádku, bude komentář vložen do závorek s hvězdičkami tj. (* pro začátek komentáře a *) pro konec komentáře.

❖ Funkce **Random**$_{\mathrm{u}}$ (např. $\mathrm{Random}_{\mathrm{u}}(a_j, b_j)$) s indexem u (*Uniform*) generuje náhodná čísla na základě rovnoměrného rozdělení na mezích intervalu $[a_j, b_j)$ v oboru reálných čísel, kde dolní mez a_j je zahrnuta do intervalu a horní mez b_j zahrnuta není. Pokud nebudou uvedeny hranice, bude generováno náhodné číslo v intervalu $[0, 1)$.

$$\mathrm{Random}_{\mathrm{u}}(a_j, b_j) \in [a_j, b_j) \subseteq \mathbb{R}, a_j \in \mathbb{R}, b_j \in \mathbb{R} \tag{3.24}$$

$$\mathrm{Random}_{\mathrm{u}}(b_j) \equiv \mathrm{Random}_{\mathrm{u}}(0, b_j) \in [0, b_j) \subseteq \mathbb{R} \tag{3.25}$$

$$\mathrm{Random}_{\mathrm{u}}(\ \ ) \equiv \mathrm{Random}_{\mathrm{u}}(0{,}1) \in [0, 1) \subseteq \mathbb{R} \tag{3.26}$$

$r \longleftarrow \mathrm{Random}_{\mathrm{u}}(a, b)$		
Funkce, jejímž výstupem je náhodné číslo v rozsahu dolní (včetně této meze) a horní meze (není prvkem tohoto intervalu) generované na základě rovnoměrného rozdělení.		
Vstup:	a:	Dolní mez intervalu pro generování náhodných čísel
Vstup:	b:	Horní mez intervalu pro generování náhodných čísel
Výstup:	r:	Náhodné číslo na základě rovnoměrného rozdělení

```
1   begin
2     Randomize;
3     result ⟵ a + Random_u(  ) * (b − a);
4   end;
```

Algoritmus 3-1 Generování náhodného čísla v rozsahu mezí na základě rovnoměrného rozdělení - $\mathrm{Random}_{\mathrm{u}}$

Všechna náhodná čísla v tomto intervalu mají stejnou pravděpodobnost, že budou (mohou být) vygenerována, hustota pravděpodobnosti je rovna $\frac{1}{b-a}$.

Druhou možností generování bodů v sousedství je za pomoci symetrického normálního rozdělení.

❖ Funkce **Random**$_{\mathrm{n}}$ (např. $\mathrm{Random}_{\mathrm{n}}(\mu, \sigma)$) s indexem n (*Normal*) generuje náhodná čísla na základě normálního (Gaussova) rozdělení se střední hodnotou μ a směrodatnou odchylkou σ. Pokud nebudou definovány hranice, bude navráceno náhodné číslo v normovaném rozdělení tj. se střední hodnotou $\mu = 0$ a se směrodatnou odchylkou $\sigma = 1$.

$$\mathrm{Random}_{\mathrm{n}}(\mu, \sigma) \sim N(\mu, \sigma) \tag{3.27}$$

$$\mathrm{Random}_{\mathrm{n}}(\ \ ) \equiv \mathrm{Random}_{\mathrm{n}}(0{,}1)$$

3.3.3.2 Penalizační funkce (Penalty Function)

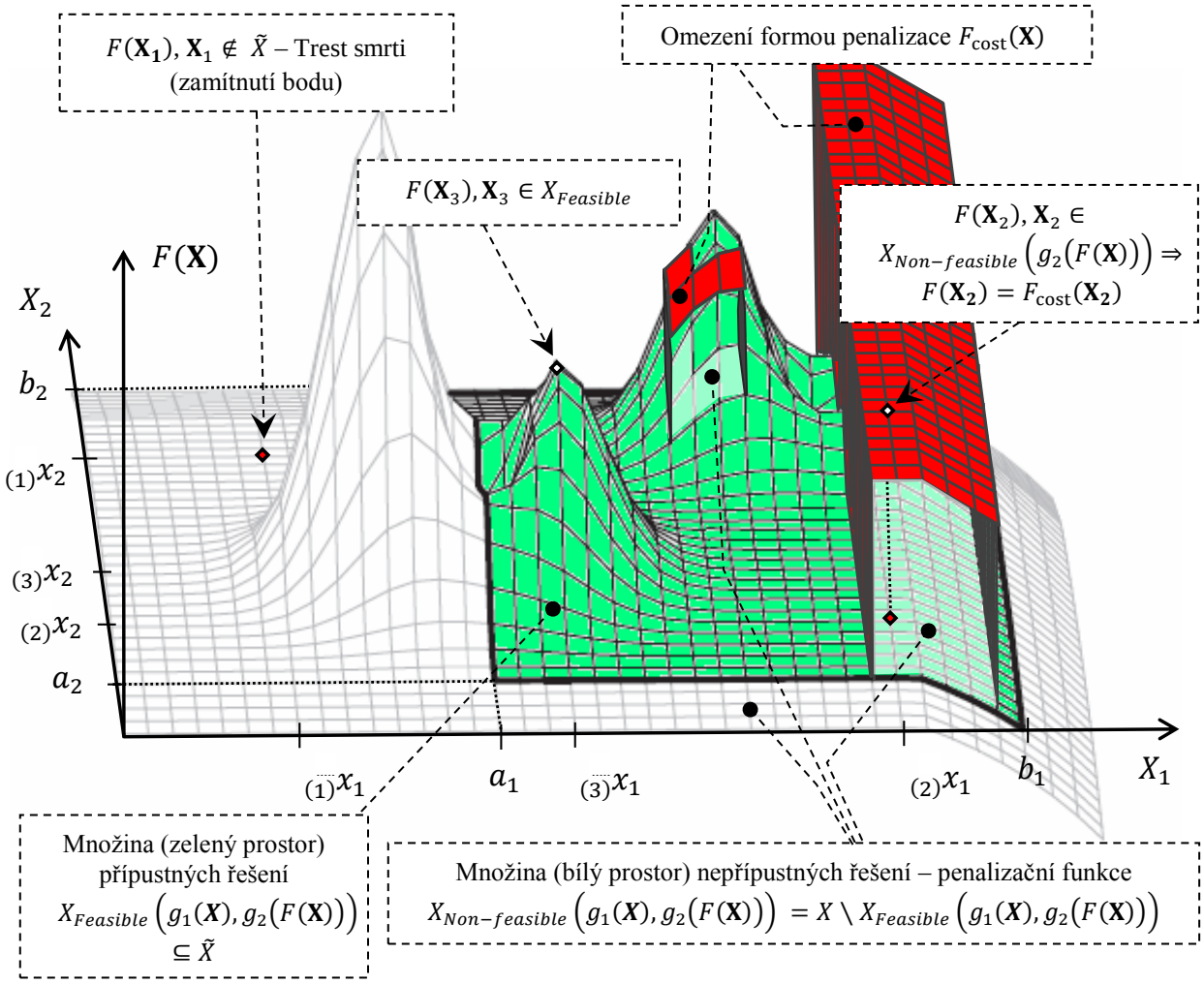
Pokud se nahlíží na prohledávaný prostor možných řešení účelové funkce jako na „životní prostředí“ prvků (viz Obr. 3-6 Příklady různého druhu omezení – červené oblasti), pak penalizace má efekt „znepříjemnění“ pobytu prvků v oblastech nepřípustných řešení – tzv. penalizovaných oblastech prostoru řešení.

Z geometrického hlediska si lze toto „znepříjemnění“ představit jako lokální deformaci hyperpochy směrem k opačnému extrému, než je hledán. Ostrůvková přípustná řešení jsou obvykle obtížně analyticky řešitelná a navíc některá omezení mohou být ve vzájemném konfliktu, což obvykle vede k tomu, že optimální řešení neexistuje. Velmi výraznou výhodou penalizačního přístupu je, že prostor možných řešení zůstává spojitý. Mnohonásobná omezení (*Hard-Constraints*) obvykle rozdělí prohledávaný prostor do izolovaných ostrůvků vhodných řešení. Tato diskontinuita mimo jiné může vytvářet různé lokální extrémy na hranicích těchto „ostrůvků“ možných řešení. Z těchto a dalších důvodů je penalizační přístup považován za mnohem výhodnější navzdory faktu, že mnoho tradičních metod optimalizace je schopno pracovat jen s *Hard-Constraints* přístupem. Na základě mnoha provedených experimentů [11] a z nich plynoucích zkušeností lze s určitostí konstatovat, že pro optimalizaci pomocí evolučních algoritmů je penalizační přístup mnohem přirozenější.

Evoluční algoritmy a zvláště diferenciální evoluce (např. SOMA – SamoOrganizující se Migrační Algoritmus) jsou i v takových případech schopny najít alespoň jedno či více sub-optimálních řešení. Penalizační funkce je zejména použitelná pro tzv. slabá omezení vyjadřující poměr počtu všech možností a počtu přípustných řešení. Pokud není tento poměr příliš nepříznivý, lze tento typ opravného algoritmu použít. [9]

Vskutku silná omezení můžeme nalézt spíše v oblasti kombinatorických optimalizací. Volba pokutové funkce bude však vždy záviset na dané úloze. Je spíše otázkou citu, či experimentu, zda pokuta bude konstantní, nebo zda bude záviset na vhodně navržené míře porušení omezovacích podmínek. [12]

Následující obrázek (viz Obr. 3-6) ilustruje příklad dvou druhů omezení – trest smrti a penalizace. V příkladu je hledáno minimum účelové funkce, tudíž abychom pomocí penalizace znepříjemnily pobyt prvkům v oblastech nepřípustných řešení, je k jejich hodnotě účelové funkce přičtena konstanta. Kvalita prvku, která je posuzována hodnotou účelové funkce, je tímto způsobem degradována.



Obr. 3-6 Příklady různého druhu omezení

3.3.3.3 *Opravné algoritmy (Repair Algorithms) a speciální operátory*

Při použití opravných algoritmů nebo speciálních operátorů optimalizační algoritmus pracuje pouze nad množinou přípustných řešení a jedině prvky náležící této množině mají příležitost účastnit se optimalizačního procesu. Následně jsou jednotlivé prvky ohodnocovány přímo použitím účelové funkce, bez nutnosti používat jakoukoliv formu penalizace. Aby bylo možné tohoto stavu dosáhnout, je nezbytné použít speciální techniky, které zajistí přípustnost všech nově generovaných prvků. V případu evolučních algoritmů existují v podstatě dva odlišné způsoby, jak dosáhnout stavu, aby prvek (v případě evolučních algoritmů jedinec) náležel množině přípustných řešení.

- Jednak lze použít tradiční genetické operátory, které ovšem pracují nad celým prostorem prohledávání $\tilde{X}$, a proto je nutné jakákoliv nepřípustná individua, která nám v procesu rekombinace vzniknou, modifikovat vhodným opravným algoritmem tak, aby výsledné individuum bylo přípustným řešením.
- Druhý způsob je založen na myšlence využití speciálních operátorů, u nichž je garantováno, že jejich použitím na přípustné individuum čí více individuí nemůže jako produkt příslušné operace vzniknout individuum nepřípustné. [13]

V případě, kdy vygenerovaný prvek pomocí optimalizačního algoritmu leží mimo povolené meze (mimo definovaných hranic prohledávaného prostoru) je nutné zajistit, aby hodnoty rozhodovacích proměnných prvku po jejich přepočítání ležely uvnitř $\tilde{X}$. Funkce **New** zajišťuje nahrazení (viz Algoritmus 3-2 Transformace souřadnic prvku mimo meze) všech nepřípustných souřadnic rozhodovacích proměnných prvku $\mathbf{X}$ ležících mimo prohledávaný prostor, nově vygenerovaným náhodným číslem na základě rovnoměrného rozdělení v rozsahu

mezí jednotlivého prohledávaného intervalu (viz Algoritmus 3-2, řádek 5 – dále bude uváděno jen číslo řádku). V algoritmu je zapotřebí přistupovat k hodnotám souřadnic rozhodovacích proměnných prvku. Protože záleží na pořadí jednotlivých složek – souřadnic rozhodovacích proměnných - v prvku, lze prvek vnímat jako seznam. Přístup k jednotlivým prvkům seznamu je pomocí indexu v hranatých závorkách (složky uvnitř seznamu budou indexovány od počátečního indexu = 0 až do posledního indexu = počet rozhodovacích proměnných – 1). Tím vytvoříme pořadí hodnot jednotlivých rozhodovacích proměnných (řádek 3). Funkce Length navrací velikost seznamu – v tomto případě prvku $\mathbf{X}$.

$\mathbf{X}_{\mathrm{New}} \longleftarrow \mathrm{New}(\mathbf{X}, A, B)$	
Funkce, jejímž výstupem je prvek, který vznikl transformací souřadnic rozhodovacích proměnných vstupního prvku, které ležely mimo rozsah intervalu jednotlivé rozhodovací proměnné. Jednotlivé nepřípustné souřadnice jsou nahrazeny vygenerovaným náhodným číslem na základě rovnoměrného rozdělení v rozsahu mezí jednotlivého prohledávaného intervalu.	
Vstup:	$\mathbf{X}$: Prvek, který obsahuje souřadnice rozhodovacích proměnných mimo prohledávaný interval rozhodovací proměnné
Vstup:	A: Seznam obsahující dolní meze intervalů pro jednotlivé rozhodovací proměnné
Vstup:	B: Seznam obsahující horní meze intervalů pro jednotlivé rozhodovací proměnné
Data:	j: Čítač – vyjadřuje index rozhodovací proměnné
Výstup:	$\mathbf{X}_{\mathrm{New}}$: Nově transformovaný prvek

```
1   begin
2     X_New ⟵ X;
3     for j ⟵ 0 to Length(X) − 1 do
      (*Length navrací délku prvku - seznamu, tj. počet v něm uložených souřadnic rozhodovacích
      proměnných; j=index rozhodovací proměnné*)
4       if (X[j] < A[j]) or (X[j] > B[j]) then
5         X_New[j] ⟵ Random_u(A[j], B[j]);
6     result ⟵ X_New[j];
7   end;
```

Algoritmus 3-2 Transformace souřadnic prvku mimo meze

Dalším způsobem jak „opravit“ souřadnice rozhodovacích proměnných prvků, které jsou mimo interval prohledávaného prostoru, je použití opravného algoritmu tzv. **zrcadlení** - **perturbace** (*Perturbation*). [14]

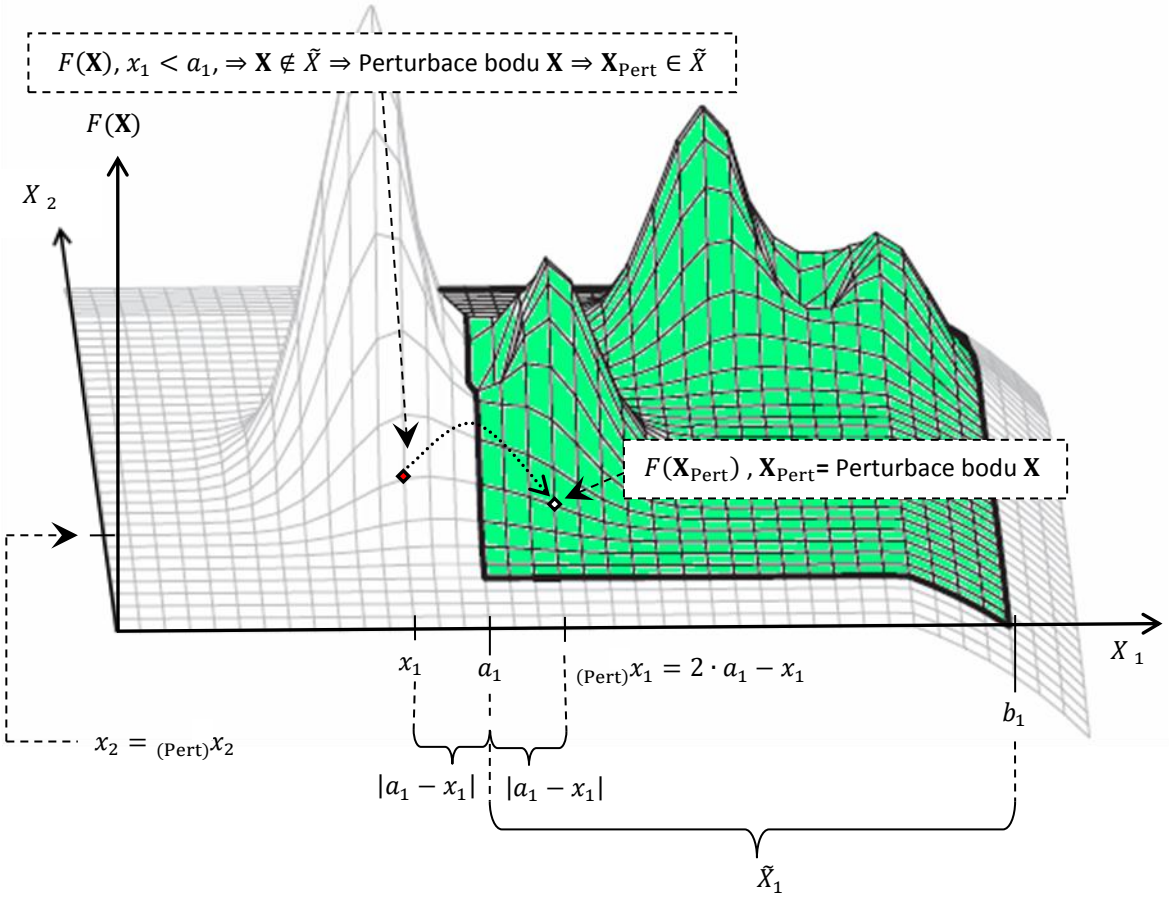
Pojem perturbace je také užíván v algoritmu simulovaného žíhání, kde na rozdíl od předchozího způsobu opravy souřadnic rozhodovacích proměnných, představuje náhodnou výměnu dvou složek prvku. Pokud prvek obsahuje n reálných na sobě nezávislých položek, modifikuje se náhodná položka prvku přičtením nebo odečtením náhodné hodnoty k hodnotě položky.[3]

Vraťme se ale k prvnímu způsobu perturbace. Souřadnici, která leží mimo prohledávaný interval rozhodovací proměnné, překlopíme dovnitř prohledávaného prostoru $\tilde{X}$ kolem příslušné hrany prohledávaného prostoru. Znamená to, že pokud je souřadnice menší než dolní mez prohledávaného intervalu j-té rozhodovací proměnné $\tilde{X}_j$, je tato souřadnice překlopena o vzdálenost této souřadnice k dolní mezi dovnitř prohledávaného prostoru. A naopak - pokud je souřadnice větší než horní mez prohledávaného intervalu j-té rozhodovací proměnné $\tilde{X}_j$, je tato souřadnice překlopena o vzdálenost horní mezi k této souřadnice dovnitř prohledávaného prostoru. Princip perturbace je patrný na příkladu v následujícím obrázku (viz Obr. 3-7 Příklad principu perturbace prvku).

[3] http://feldocuments.googlecode.com/svn-history/r127/trunk/NS/msi.pdf

V grafu účelové funkce je znázorněn prvek (bod), který nenáleží prohledávanému prostoru:

$$\mathbf{X} \notin \tilde{X} \tag{3.28}$$



Obr. 3-7 Příklad principu perturbace prvku

Prvek $\mathbf{X}$ nenáleží prohledávanému prostoru, protože jeho první složka, tj. hodnota rozhodovací proměnné, leží mimo prohledávaný interval první rozhodovací proměnné:

$$\mathbf{X} = [x_1, x_2], x_1 : x_1 < a_1 \Rightarrow x_1 \notin \tilde{X}_1 \tag{3.29}$$

kde:

- $\mathbf{X}$ … Prvek – bod v grafu účelové funkce.
- x_1 … Hodnota rozhodovací první rozhodovací proměnné – souřadnice na první ose v grafu účelové funkce.
- a_1 … Dolní mez prohledávaného intervalu první rozhodovací proměnné.
- $\tilde{X}_1$ … Prohledávaný interval první rozhodovací proměnné.

Perturbaci popisuje následující algoritmus. V algoritmu bod v grafu účelové funkce představuje seznam nastavených hodnot jednotlivých rozhodovacích proměnných (viz Algoritmus 3-3, řádek 3 – dále bude uváděno jen číslo řádku).

Princip zrcadlení hodnoty rozhodovací proměnné prvku, tj. zrcadlení hodnoty rozhodovací proměnné kolem dolní meze (řádek 6) nebo kolem horní meze (řádek 8) je uplatňován tak dlouho, dokud hodnota rozhodovací proměnné neleží uvnitř prohledávaného intervalu rozhodovací proměnné (řádek 4).

- Funkce **Perturbation** (např. $\mathbf{X}_{\mathrm{Pert}} \longleftarrow \mathrm{Perturbation}(\mathbf{X}, A, B)$) velmi jednoduchým způsob garantující „návrat" prvku, který tyto hranice překročí, je zachycen pomocí následujícího algoritmu (Algoritmus 3-3 Perturbace souřadnic prvku mimo meze): [14]

$\mathbf{X}_{\mathrm{Pert}} \longleftarrow \mathrm{Perturbation}(\mathbf{X}, A, B)$	
Funkce, jejímž výstupem je prvek, který vznikl perturbací – zrcadlením - souřadnic rozhodovacích proměnných vstupního prvku, které ležely mimo rozsah intervalu jednotlivé rozhodovací proměnné.	
Vstup:	$\mathbf{X}$: Prvek, který obsahuje souřadnice rozhodovacích proměnných mimo prohledávaný interval rozhodovací proměnné
Vstup:	A: Seznam obsahující dolní meze intervalů pro jednotlivé rozhodovací proměnné
Vstup:	B: Seznam obsahující horní meze intervalů pro jednotlivé rozhodovací proměnné
Data:	j: Čítač – vyjadřuje index rozhodovací proměnné
Výstup:	$\mathbf{X}_{\mathrm{Pert}}$: Prvek vzniklý perturbací původního prvku.

```
1   begin
2     X_Pert ⟵ X;
3     for j ⟵ 0 to Length(X) − 1 do
      //indexování rozhodovacích proměnných - dimenzí
4         while (X[j] < A[j]) or (X[j] > B[j]) do
5             if (X[j] < A[j]) then
6                 X_Pert[j] ⟵ 2 · A[j] − X[j]
7             else if (X[j] > B[j]) then
8                         X_Pert[j] ⟵ 2 · B[j] − X[j];
9     result ⟵ X_Pert[j];
10  end;
```

Algoritmus 3-3 Perturbace souřadnic prvku mimo meze [14]

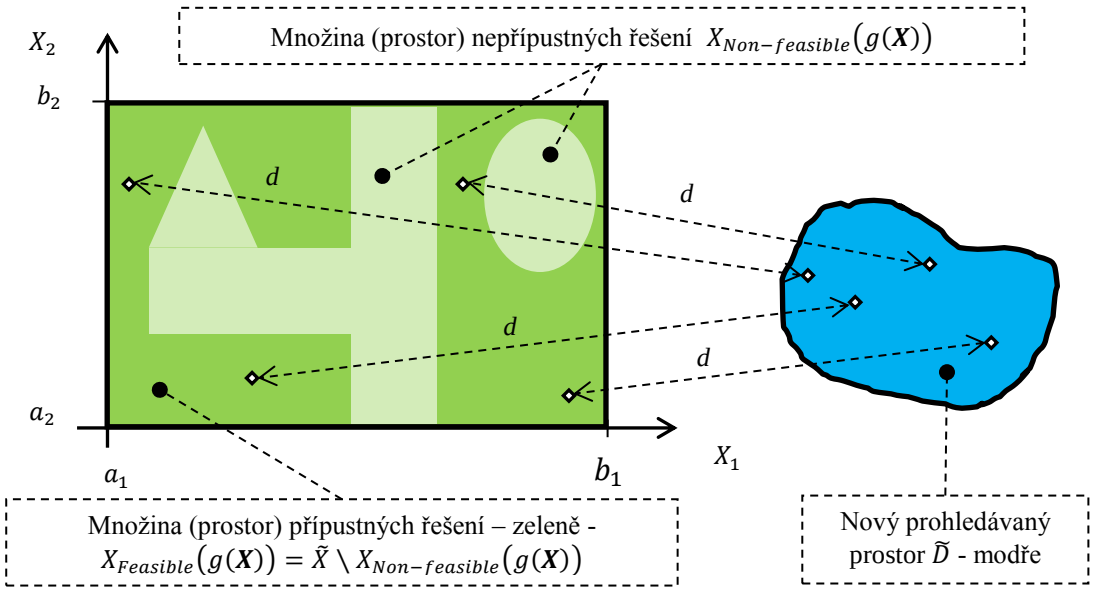
3.3.3.4 Dekodéry (Decoders)

Dekodéry představují další alternativu, jak pracovat s omezujícími podmínkami. Dekodéry jsou využívány zejména v oblasti genetických algoritmů. U genetických algoritmů je prvek zakódován do tzv. chromozomu, který se skládá z jednotlivých částí, které reprezentují hodnoty rozhodovacích proměnných umístěných v prvku a jsou vyjádřeny pomocí binárního kódování - soustavou jedniček a nul - nebo pomocí jiné soustavy. Při použití dekodéru dochází k tomu, že chromozom není chápán jako zakódované řešení, ale spíše jako informace, na jejímž základě lze přípustné řešení sestrojit. Namísto prohledávaného prostoru či jeho podmnožin je možné použitím vhodné reprezentace vytvořit zcela jiný prostor prohledávání a dekodér potom slouží jako transformace mezi jednotlivými prvky v tomto prostoru a jejich obrazem v množině přípustných řešení (viz Obr. 3-8):

$$d: \widetilde{D} \rightarrow X_{Feasible}\big(g(\boldsymbol{X})\big) \tag{3.30}$$

kde:

- d … Dekodér - transformace.
- $\widetilde{D}$ … Nový prohledávaný prostor.
- $\mapsto$ … Zobrazení - předpis jak přiřazovat prvkům nějaké množiny jednoznačně prvky obecně jiné množiny.
- $X_{Feasible}\big(g(\boldsymbol{X})\big)$ … Množina (prostor) přípustných řešení vyhovující podmínce.
- $\big(g(\boldsymbol{X})\big)$ … Omezující složená podmínka – soustava podmínek.



Obr. 3-8 Příklad transformace mezi prohledávaným prostorem a novým prohledávaným prostorem [13]

Je zřejmé, že evoluční algoritmus pracuje nad novým prostorem, jehož individua jsou pomocí dekodéru zobrazena na množinu přípustných řešení, čímž dochází k nepřímému prohledávání této množiny. Dekódovaná individua jsou přípustnými řešeními daného problému, a proto lze k jejich ohodnocení s výhodou použít účelové funkce bez jakýchkoliv dalších úprav. Protože takto byl problém splnění omezujících podmínek převeden na schopnost dekodéru najít k libovolnému prvku z nového prohledávaného prostoru přípustné řešení z původního prohledávaného prostoru, je možné, aby genetický algoritmus pracující nad novým prostorem využíval zcela standardní genetické operátory. Obecná myšlenka, na níž je celý přístup použití dekodéru založen, je relativně jednoduchá a i v tomto případě je zcela evidentní, že při vlastním návrhu dekodéru bude opět nutné využít podrobné znalosti o konkrétním problému a příslušných omezujících podmínkách. První nutnou podmínkou použitelnosti konkrétního dekodéru je efektivní algoritmus, kterým je tento dekodér realizován. Protože bude třeba dekódovat v každé iteraci evolučního procesu velké množství prvků, je nezbytné hledat takový algoritmus, jehož časová náročnost je co nejnižší. Druhým požadavkem je lokálnost dekodéru, která se projevuje tím, že malá změna prvku z nového prostoru se projeví malou změnou odpovídajícího přípustného řešení z původního prostoru. Pokud bude princip lokálnosti porušen nevhodným návrhem dekodéru, potom evoluční proces ztratí schopnost prohledávat okolí slibných individuí a změní se na ryze náhodné prohledávání. [13]

3.4 Seznam

Protože v dalších kapitolách budou vysvětlovány principy metod globální optimalizace uplatňované na prvky, které budou umísťovány do seznamů (záleží na pořadí prvků), jsou v této kapitole stručně popsány jednotlivé metody práce se seznamy. Metody jsou definovány za účelem zpřehlednění jednotlivých optimalizačních algoritmů. U metod práce se seznamy je uveden matematický popis a stručný popis principu těchto metod. Matematické popisy metod, jsou převzaty z literatury - viz [10]. Některé z těchto metod byly modifikovány pro potřeby jednotlivých optimalizačních algoritmů uvedených v disertační práci.

Důležitou součástí této disertační práce je **příloha disertační práce**, kde jsou podrobně popsány principy optimalizačních algoritmů, vhodná nastavení parametrů optimalizačních algoritmů, testované simulační modely a také následující metody práce se seznamy (viz **příloha** - kapitola 1 Seznam).

Seznamy (jedná se o uspořádanou množinu prvků) jsou abstraktní datové typy, které mohou obsahovat uspořádanou konečnou posloupnost prvků stejného datového typu. Seznam může být považován za speciální abstraktní datový typ datového záznamu (seznam může obsahovat stejný prvek i vícekrát). Seznamy mohou být

objekty jako např. tabulky atd., proto se s nimi pracuje na základě metod uplatňovaných na daný objekt. Přístup k jednotlivým prvkům seznamu je pomocí indexu v hranatých závorkách, kde první index má hodnotu 0 a poslední index má hodnotu $n-1$, kde *n* značí počet prvků v seznamu.

Následující metody umožňují přidání prvku do seznamu, odstranění prvku ze seznamů, třídění seznamů nebo vyhledávání uvnitř seznamů atd.:

- **Length** ($n = \mathrm{Length}(L)$) funkce která navrací délku seznamu L tj. počet prvků v seznamu.
- **CreateList** ($L = \mathrm{CreateList}(n, x)$) funkce pro vytvoření nového seznamu L o délce *n*, naplněný prvkem x. Jestliže je vytvořen seznam o délce 0, parametr x může být vynechán.
- **InsertListItem** ($M = \mathrm{InsertListItem}(L, i, x)$) funkce vytvoření nového seznamu M pomocí vložení jednoho prvku *x* do seznamu L s indexem *i* ($\forall i: 0 \leq i \leq \mathrm{Length}(L)$). Tím se posunou všechny prvky seznamu umístěné za tímto indexem o jednu pozici doprava.
- **AddListItem** ($M = \mathrm{AddListItem}(L, x)$) funkce pro vložení jednoho prvku na konec seznamu.
- **AppendList** ($M = \mathrm{AppendList}(L_1, L_2)$) funkce pro vytvoření nového seznamu *M*, který vznikne přidáním všech prvků ze seznamu L_2 do seznamu L_1. Rekurzivně lze definovat:
- **DeleteListItem** ($M = \mathrm{DeleteListItem}(L, i)$) funkce pro vytvoření nového seznamu *M* pomocí odstranění prvku *x* na pozici *i* ($\forall i: 0 \leq i < \mathrm{Length}(L) - 1$) ze seznamu *L* ($\mathrm{Length}(L) \geq i + 1$).
- **DeleteListRange** ($M = \mathrm{DeleteListRange}(L, i, c)$) funkce pro vytvoření nového seznamu *M* pomocí odstranění *c* prvků začínající indexem *i* ($\forall i: 0 \leq i < \mathrm{Length}(L)$ ze seznamu *L* ($\mathrm{Length}(L) \geq i + c$).
- **CountItemOccurrences** ($y = \mathrm{CountItemOccurrences}(x, L)$) funkce, která vrací počet výskytů prvku *x* v seznamu *L*.
- **SubList** ($M = \mathrm{SubList}(L, i, c)$) funkce vrací nový seznam *M* vyjmutých *c* prvků ze seznamu *L* začínající indexem *i*.
- **CF** - komparační (porovnávací) funkce – ($\mathrm{CF}(l_1, l_2)$), která vrací zápornou hodnotu jestli $l_1 < l_2$, kladnou hodnotu když $l_1 > l_2$ a hodnotu 0, pokud $l_1 = l_2$. Funkce je využita pro párové porovnání dvou prvků ze seznamu při třídění. Pokud bude u komparační funkce použito indexování, znamená to, že prvky byly porovnány na základě hodnot funkce vypočtené pro každý prvek např. $\mathrm{CF}_{F(\mathbf{X})}(l_1, l_2)$ ... prvky byly porovnány na základě hodnot účelové funkce u každého prvku.
- **Sort** $M = \mathrm{Sort}_{\mathrm{a}}(L, \mathrm{CF})$ a $M = \mathrm{Sort}_{\mathrm{d}}(L, \mathrm{CF})$) často je užitečné používat setříděný seznam, proto je zde definována funkce, která třídí seznam *L* podle pořadí buď vzestupně – ascending - $M = \mathrm{Sort}_{\mathrm{a}}(L, \mathrm{CF})$ nebo sestupně – descending - $M = \mathrm{Sort}_{\mathrm{d}}(L, \mathrm{CF})$ pomocí komparační funkce.
- $\mathbf{Search}_{\mathbf{u}}(\boldsymbol{l}, \boldsymbol{L})$ **-** hledání prvku *l* v nesetříděném seznamu *L* znamená projít všechny prvky do té doby, dokud není prvek nalezen, nebo dokud se nenarazí na konec seznamu.
- **RemoveListItem** ($M = \mathrm{RemoveListItem}(L, x)$) nalezení prvního výskytu prvku *x* v seznamu *L* pomocí vhodného algoritmu a tento prvek bude vymazán (navrácení nového seznamu *M*).

3.5 Globální optimalizace

Globální optimalizace se zabývá optimalizací jednoho nebo více kritérií, která mohou být navzájem protichůdná. Tato kritéria jsou vyjádřena jako množina jednotlivých účelových (kriteriálních, objektivních, cílových) funkcí. Problematikou optimalizace více účelových funkcí se zabývá disciplína operačního výzkumu – vícekriteriální rozhodování. Tato disciplína ale neposkytuje návod, jak v takových úlohách jednoznačně postupovat. Záleží tedy na zkušenostech a úvaze analytika, pro kterou metodu vícekriteriálního hodnocení se rozhodne.

Souhrnně se dá říci, že globální optimalizace se snaží o nalezení optimální konfigurace systému. V případě technické optimalizace to tedy znamená nalezení takového nastavení hodnot rozhodovacích proměnných simulačního modelu (hodnot vstupních parametrů simulačního modelu), aby bylo zajištěno nejlepší uspořádání nebo chování simulačního modelu vzhledem ke zvolenému cíli (který reprezentuje účelová funkce) za respektování nadefinovaných omezujících podmínek. Argumenty účelové funkce (bývají to většinou odezvy – výstup/y ze simulačního modelu) mohou být např. celočíselné, či reálné. Hodnoty účelové funkce se získávají z odezev získaných po proběhnutí simulačního experimentu na diskrétním simulačním modelu, a proto nemůžeme v simulačních úlohách použít analytická řešení pro nalezení extrému funkce (tato funkce je složitá,

např. multimodální, nespojitá, definovaná jen v diskrétních hodnotách, nabývá náhodných hodnot, s omezujícími podmínkami atd.). Jsme proto odkázáni na numerické (heuristické) postupy, které v přijatelném čase naleznou optimální řešení.

Ve většině případů optimalizační metoda hledá globální maximum nebo minimum účelové funkce. Úlohu nalezení globálního minima na definičním oboru (v našem případě je prostor omezen na prostor prohledávání) můžeme formulovat takto:

$$\breve{\mathbf{X}} = \operatorname{argmin}_{\mathbf{X}\in\tilde{X}} F(\mathbf{X}) = \left\{\breve{\mathbf{X}} \in \tilde{X}: F\left(\breve{\mathbf{X}}\right) \leq F(\mathbf{X}) \forall \mathbf{X} \in \tilde{X}\right\} \tag{3.31}$$

kde:

- $\breve{\mathbf{X}}$ … Hodnota rozhodovací proměnné, pro kterou nastává globální minimum účelové funkce.
- $F(\mathbf{X})$ … Účelová funkce – obor hodnot jsou reálná čísla tj. $F(\mathbf{X}) \subseteq \mathbb{R}$.
- $\tilde{X}$ … Souvislý prohledávaný prostor.

V prohledávaném prostoru rozlišujeme dva základní typy extrémů – **lokální** a **globální**. Globálním optimem je v tomto případě jedno optimum na celém prohledávaném prostoru, zatímco lokálním optimem je pouze optimum jedné vlastní podmnožiny prohledávaného prostoru.

Lokální minimum $\breve{\mathbf{X}}_\mathrm{l} \in \tilde{X}$ účelové funkce $F(\mathbf{X}) \subseteq \mathbb{R}$ je takovým bodem, že pro všechna $\mathbf{X}$ sousedící s $\breve{\mathbf{X}}_\mathrm{l}$ platí:

$$F\left(\breve{\mathbf{X}}_\mathrm{l}\right) \leq F(\mathbf{X}) \tag{3.32}$$

kde:

- $\breve{\mathbf{X}}_\mathrm{l}$ … hodnota rozhodovací proměnné, pro kterou nastává lokální minimum účelové funkce.

Jestliže platí $\tilde{X} \subseteq \mathbb{R}$, lze říci že:

$$\breve{\mathbf{X}}_\mathrm{l}: \exists e > 0: F\left(\breve{\mathbf{X}}_\mathbf{l}\right) \leq F(\mathbf{X}) \forall \mathbf{X} \in \tilde{X}, \left|\mathbf{X} - \breve{\mathbf{X}}_\mathrm{l}\right| < e \tag{3.33}$$

kde:

- e… Hranice okolí bodu lokálního minima.

Globální maximum $\hat{\mathbf{X}} \in \tilde{X}$ účelové funkce $F(\mathbf{X}) \subseteq \mathbb{R}$ je takovým bodem, že platí:

$$F\left(\hat{\mathbf{X}}\right) \geq F(\mathbf{X}) \forall \mathbf{X} \in \tilde{X} \tag{3.34}$$

Lokální maximum $\hat{\mathbf{X}}_\mathrm{l} \in \tilde{X}$ účelové funkce $F(\mathbf{X}) \subseteq \mathbb{R}$ je takovým bodem, že pro všechna $\mathbf{X}$ sousedící s $\hat{\mathbf{X}}_\mathrm{l}$ (tj. body v jeho okolí) platí:

$$F\left(\hat{\mathbf{X}}_\mathrm{l}\right) \geq F(\mathbf{X}) \tag{3.35}$$

Jestliže platí $\tilde{X} \subseteq \mathbb{R}$, lze říci že:

$$\hat{\mathbf{X}}_\mathrm{l}: \exists e > 0: F\left(\hat{\mathbf{X}}_\mathrm{l}\right) \geq F(\mathbf{X}) \forall \mathbf{X} \in \tilde{X}, \left|\mathbf{X} - \hat{\mathbf{X}}_\mathrm{l}\right| < e \tag{3.36}$$

Pokud alespoň jedna souřadnice prvku - hodnota rozhodovací proměnné je z množiny reálných čísel a optimalizační algoritmus vyhledává např. globální minimum, je nutné si uvědomit, že nalezeným globálním

minimem může být infimum (za podmínky, že prohledávaný prostor je zdola omezená množina – pro náš případ interval $\tilde{X}_j = (a_j, b_j)$, tj. nelze brát jako přípustné řešení prvek, jehož vygenerovaná souřadnice je rovna mezi). Nejmenší prvek je pak právě a_j (jde o dolní závoru a jakékoliv větší číslo již není dolní závorou – lze argumentovat podobně jako u minima). Pokud má množina minimum má i infimum tzn., že minimum je zároveň infimem. Duálním pojmem (opakem) infima je supremum. [4]

Lokální optimum $\mathbf{X}_l^* \in \tilde{X}$ účelové funkce $F(\mathbf{X}) \subseteq \mathbb{R}$ je buď lokální maximum nebo lokální minimum (nebo obě).

Lokální extrémy funkce - tedy lokální maximum a lokální minimum, bychom u funkcí umožňujících derivaci odhalili tak, že položíme první derivaci funkce rovnou nule a vypočítáme kořeny vzniklé rovnice.[5]

Globální optimum $\mathbf{X}^* \in \tilde{X}$ účelové funkce $F(\mathbf{X}) \subseteq \mathbb{R}$ je buď globální maximum, nebo globální minimum.

Při existenci omezení musí prvek reprezentující možné řešení splňovat všechna definovaná omezení:

$$\breve{\mathbf{X}} \in X_{Feasible}(g_1, g_2, \dots, g_o) \tag{3.37}$$

kde:

- $\breve{\mathbf{X}}$ …Nejlepší řešení - globální minimum.
- $X_{Feasible}$ … Množina - prostor přípustných řešení.
- g_i … i-tá funkce omezení.

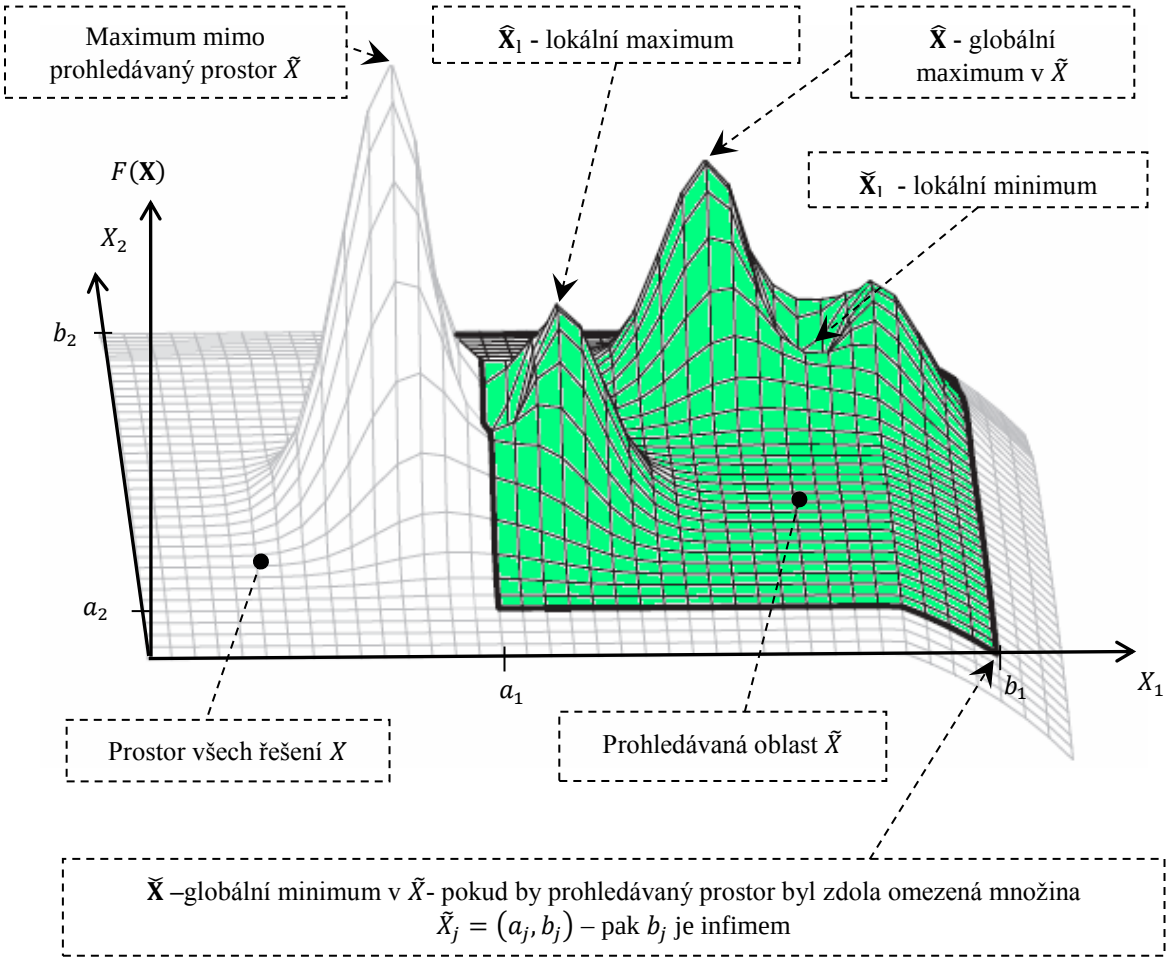
Pro globální minimum platí, že všechny ostatní prvky v množině přípustných řešení jsou horším řešením, tj. neexistuje lepší prvek v množině přípustných řešení. Jinak řečeno všechny ostatní prvky z množiny přípustných řešení jsou větší než tento optimální prvek vyjadřující globální minimum:

$$\forall \mathbf{X} \in X_{Feasible}(g_1, g_2, \dots, g_o), F\left(\breve{\mathbf{X}}\right) \leq F(\mathbf{X}) \tag{3.38}$$

Následující obrázek (viz Obr. 3-9 Příklad možných výsledků globální optimalizace účelové funkce s omezením prostoru všech řešení na prohledávaný prostor) ilustruje úlohu jednoúčelové globální optimalizace - hledání extrému účelové funkce $F(\mathbf{X})$ na prostoru $\tilde{X}$. Prohledávaný prostor reprezentují dvě rozhodovací proměnné $n = 2 \Rightarrow X = \{X_1, X_2\}, \tilde{X} = [a_1, b_1] \times [a_2, b_2]$).

[4] http://cs.wikipedia.org/wiki/Infimum

[5] http://www.aristoteles.cz/matematika/funkce/vysetrovani/lokalni-extremy-funkce-monotonnost-rostouci-klesajici.php



Obr. 3-9 Příklad možných výsledků globální optimalizace účelové funkce s omezením prostoru všech řešení na prohledávaný prostor, sousedství prvku

3.6 Problémy globální optimalizace

Základním společným problémem globální optimalizace je, jak co nejrychleji najít optimum v okolí výchozího bodu. S tím také souvisí otázka, jak nejlépe prohledat prostor, ve kterém se nacházejí možná řešení (většinou přijímáme uspokojivá řešení daného problému). Často si většinou nemůžeme dovolit prohledávat celý prostor možných (přípustných) řešení z důvodu velkých nároků na operační paměť počítače nebo na čas vynaložený na prohledávání prostoru.

Dalším praktickým problémem je, že v mnoha případech nelze jednoznačně určit, zda nejlepší nalezené řešení je lokálním nebo globálním optimem. Jinými slovy lze říci, že není jednoznačně dané, zda se zaměřit na jemnější prohledávání nalezení dosavadního optima nebo raději vyzkoušet prohledávat jiné části prostoru.

3.6.1 NP-problém

Problémy např. spojené s rozvrhováním výroby lze zařadit do třídy tzv. **nedeterministicky polynomiálních problémů – NP** (*Non-Deterministic Polynomial*). U takových problémů lze říci, že ohodnocení všech různých variant s cílem nalézt optimální varianty je zejména z časových důvodů neefektivní nebo ve většině případů nemožné.

Příklad nedeterministicky polynomiálního problému můžeme nastínit v tzv. problému obchodního cestujícího. Tento cestující má navštívit 20 různých měst a samozřejmě chce, aby navštívil města v takovém pořadí, aby součet tras jím navštívených měst byl co nejmenší (za předpokladu, že každé město navštíví jen jednou). Pokud se nad tímto případem zamyslíme, zjistíme, že kdyby obchodní cestující chtěl zkusit všechny možné kombinace

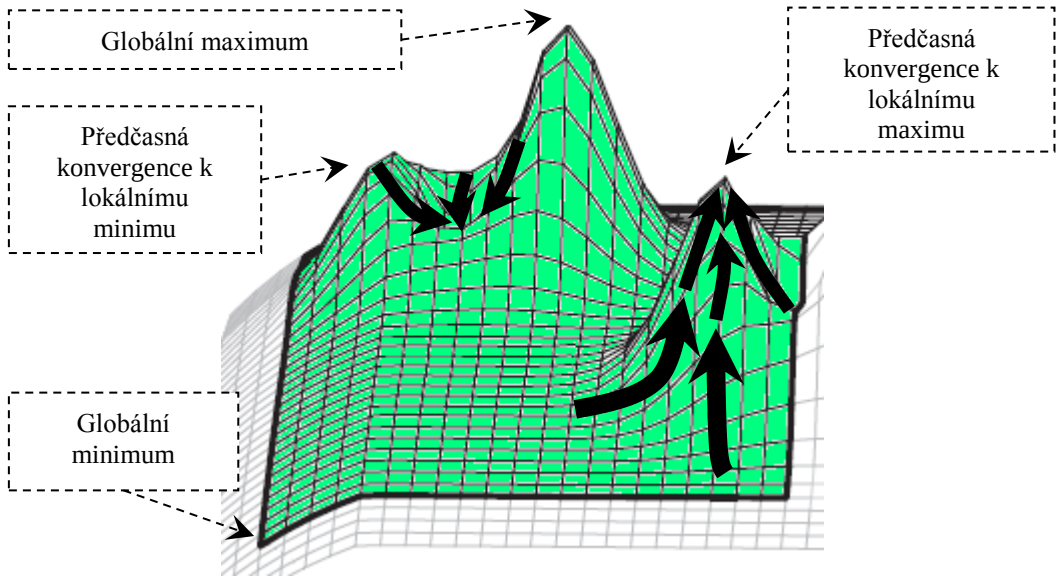
jak navštívit různá města, bez toho aniž by navštívil jedno město dvakrát, je počet všech kombinací roven 20! = $2.43290200817664 \cdot 10^{18}$ [kombinací]. Co ale dál? Je možné vyzkoušet všechny kombinace? Pokud bychom použili počítač, který by vypočítal 10 000 takových sekvencí měst za 1 sekundu, výpočet by mu trval zhruba 7714681 let.

Pro řešení takových úloh se často používají heuristické optimalizační metody, které sice nezaručují poskytnutí globálního optima jako přesného výsledku, ale jsou schopny v přijatelném čase poskytnout dobré řešení.

3.6.2 Konvergence k lokálnímu optimu

Globální optimalizační proces může **předčasně** konvergovat k lokálnímu optimu. To se stává v případech, kdy již nejsme dále schopni prozkoumávat další části prostoru řešení (je aktuálně prohledávána jen určitá oblast) a zároveň existuje další oblast prostoru (která není dosud známa) obsahující lepší řešení vzhledem k aktuálně nalezenému řešení.

Na obrázku (viz Obr. 3-10 Příklad předčasné konvergence optimalizačního algoritmu v prohledávaném prostoru) je ilustrována předčasná konvergence optimalizačního algoritmu k lokálnímu optimu. Optimalizační algoritmus, než dojde k dobrému výsledku, by měl projít (ideálně všemi) lokálními optimy. Může se ale stát, že uvázne v určité oblasti, kde se zacyklí a nemůže pokračovat v hledání všech dalších kandidátů na optimum („myslí si“, že našel oblast, kde se nalézá globální optimum a tu se snaží jemněji prohledávat).



Obr. 3-10 Příklad předčasné konvergence optimalizačního algoritmu v prohledávaném prostoru [15]

Je mnoho rysů a parametrů nastavení optimalizačních algoritmů, které ovlivňují konvergenci. Jedním z takových faktorů je **adaptivní chování** (*Self-Adaptation*), které může konvergenci ovlivnit jak pozitivně, tak i negativně. Dalším faktorem jsou operace, které vytváří nové řešení na základě stávajících.

Všechny optimalizační algoritmy musí najít kompromis mezi tzv. **diverzifikací** a **intenzifikací**: [10], [15]

- **Diverzifikace** (prozkoumávání, explorace) ve smyslu optimalizace je nacházení nových prvků uvnitř prohledávaného prostoru. Vzhledem k tomu, že v některých případech jsme limitováni pamětí počítače, musíme se občas vzdát některých již ohodnocených prvků v přesvědčení, že nalezneme lepší řešení (to platí zejména v případech, kdy se optimalizační algoritmus pohybuje v definičním oboru

reálných čísel). Bohužel na druhou stranu tento krok vede k snižování výkonnosti, až do té doby, dokud není nalezeno nové dobré řešení, což nikdy není zaručeno.

- **Intenzifikace** (zužitkování, exploatace) znamená snahu zlepšovat dosavadní známé řešení pomocí malých změn, které vedou k novým podobným zlepšeným prvkům. Touto cestou lze také dosáhnout lepší výkonnosti.

Téměř všechny optimalizační strategie mohou být buď použity pro zvyšování diverzifikace, nebo intenzifikace.

Operace, které vytvářejí nové prvky z existujících prvků, mají velký dopad na rychlost konvergence a diverzitu prvků. Mutace u algoritmů evoluce (ve smyslu intenzifikace) může například zlepšovat existující řešení tím, že na prvek budou uplatňovány pouze malé změny. Nevýhoda takového přístupu je, že algoritmus pravděpodobně neprozkoumá prvky ve vzdálenějších oblastech, které by mohly nabídnout lepší řešení problému.

Při selekci v oblasti evolučních algoritmů vybíráme množinu prvků ze seznamu kandidátů, u kterých proběhne reprodukce. V závislosti na typu selekce a typu reprodukce se tito jedinci mohou vrátit buď v podobě malé skupiny obsahující nejlepší prvky, nebo v širokém rozsahu existujících kandidátů řešení. Stejný princip platí pro oklešťování, při kterém se ořezává množina dobrých řešení v případě, že je moc veliká.

Algoritmy podporující intenzifikaci mají sice rychlou konvergenci, ale vystavují se riziku, že nenaleznou optimální řešení a možná uvíznou v lokálním optimu. Naproti tomu algoritmy, které provádí nadměrnou diverzifikaci, mohou naleznout globální optimum. Cenou za použití tohoto přístupu je, že nalezení optima bude trvat mnohem déle. Dobrým příkladem takového dilema je algoritmus simulovaného žíhání. Ten je často modifikován na simulované ochlazování (*Simulated Quenching*), který upřednostňuje intenzifikaci, ale ztrácí garanci konvergence k optimu. Zachování diverzity je hlavním zájmem optimalizace, protože její ztráta může vést k předčasné konvergenci k lokálnímu optimu. [10]

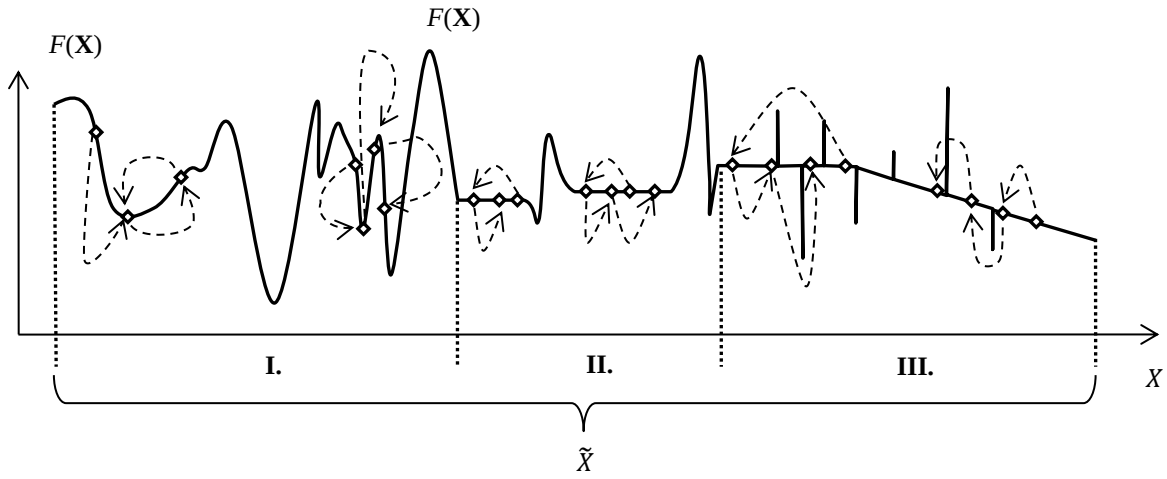
3.6.3 Průběh účelové funkce

Základní vliv na úspěšnost algoritmu při hledání globálního optima má průběh (povrch) účelové funkce. Problémem jsou zejména multimodální funkce, kde zůstává otázkou, zda optimalizační algoritmus narazil na lokální nebo globální optimum. Někdy totiž i malé změny předchozího prvku provedené za účelem generování nového prvku (např. nepatrný přírůstek přičtený k souřadnicím rozhodovacích proměnných předchozího prvku) mohou vést k velkým změnám v hodnotách účelové funkce. Pro optimalizační algoritmus je proto těžké rozhodnout, kterou oblast prostoru možných řešení prozkoumávat.

Takový případ je zachycen na dalším obrázku (viz Obr. 3-11 Příklady různých typů povrchu účelové funkce), kdy se optimalizační algoritmus snaží nalézt globální minimum účelové funkce $F(\mathbf{X})$ v jednodimenzionálním prohledávaném prostoru: [10], [15]

- Úsek č. I:
  - V levé části grafu účelové funkce je zachycen případ, že nejlepší prvek, který je doposud znán, je obklopen prvky, které jsou horší, a algoritmus může uvíznout v lokálním extrému.
  - V pravé části grafu účelové funkce tohoto úseku je patrné, že i poměrně malé změny u špatných prvků mohou vést k nalezení lepšího lokálního optima, než které bylo dosud nalezeno.
- Úsek č. II:
  - Hladkost povrchu účelové funkce může být v optimalizačním procesu problémem, ačkoliv se to na první pohled nemusí zdát. Jestliže se doposud nalezený nejlepší prvek nachází na „rovném“ povrchu, není k dispozici žádná informace o gradientu. To znamená, že není určen směr, ve kterém optimalizační algoritmus může v hledání lepšího prvku dál pokračovat. Kromě toho každý reprodukční cyklus přináší identické „optimální“ prvky a archiv pro uchovávání nejlepších prvků může brzy přetéct. Zlepšení u nových prvků (jejich hodnot účelové funkce), u povrchu II. je nízké, proto v takových rovinných oblastech optimalizační proces degeneruje na pouhé náhodné prohledávání.

- Úsek č. III:
  - Je patrné, že tato oblast je kombinací předchozích dvou uvedených typů povrchů. Takový průběh je nejhorší variantou při hledání globálního optima. Malý pohyb prvku může zapříčinit přeskočení lokálního řešení (potažmo globálního).



Obr. 3-11 Příklady různých typů povrchu účelové funkce – hledání globálního minima účelové funkce

3.7 Obecné prvky globální optimalizace

V kapitole budou uvedeny základní prvky globální optimalizace, které využívá většina metod globální optimalizace. Takovými prvky jsou:

- Minimalizace (maximalizace)
- Iterace
- Kritérium ukončení

3.7.1 Minimalizace (maximalizace)

Základním prvkem globální optimalizace je bezesporu nalezení globálního minima na celém definičním oboru, pokud prostor není omezen prohledávaným prostorem nebo na množině přípustných řešení, pokud jsou specifikována omezení.

Algoritmus může být definován jak pro maximalizaci, tak i pro minimalizaci cíle (účelové funkce), bez toho aniž by ztratil svoji všeobecnost (univerzálnost). V oblasti optimalizace je proto běžně užíván termín minimalizační proces. Algoritmus hledající maximum dané účelové funkce $y = F(\mathbf{X})$ může být transformován na úlohu hledání minima $y = -F(\mathbf{X})$. [10]

Minimum funkce $y = F(\mathbf{X})$ je ve stejném bodě jako maximum funkce $y = -F(\mathbf{X})$ a naopak. U optimalizačních metod proto postačí algoritmy, které hledají jen maximum nebo jen minimum (pokud nemají inverzi znaménka zabudovanou uvnitř algoritmu). [1]

Tento princip je zachycen na následujícím obrázku (viz Obr. 3-12 Transformace hledání maxima účelové funkce na úlohu hledání minima účelové funkce).

Poznámka: V našem případě budou popisované algoritmy porovnávat kvalitu nově vygenerovaného prvku $\mathbf{X}$ pomocí hodnoty účelové funkce daného prvku tj. $F(\mathbf{X})$. Hodnota účelové funkce bude získána na základě informací z proběhnutého simulačního experimentu – vyhodnocení simulačního modelu s konkrétním nastavením hodnot rozhodovacích proměnných – vstupních parametrů simulačního modelu. Takové algoritmy mohou být definovány jak pro maximalizaci tak minimalizaci cíle (nějaké funkce, hodnoty prvků v seznamu atd.), bez toho aniž by ztratily svoji univerzálnost. V dalším popisu optimalizačních metod proto budeme obecně používat **minimalizaci** účelové funkce.

- **CF -** komparační funkce - ($l \longleftarrow \mathrm{CF}_{F(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2)$), jejíž výstup poskytuje informaci, zda je hodnota účelové funkce prvku $\mathbf{X}_1$ tj. $F(\mathbf{X}_1)$ lepší, horší nebo stejná, než hodnota účelové funkce druhého prvku $F(\mathbf{X}_2)$. Pokud tedy v případě minimalizace účelové funkce bude $F(\mathbf{X}_1) < F(\mathbf{X}_2)$, tzn. první prvek je lepší než druhý (viz Algoritmus 3-4, řádek 2 – dále bude uváděno jen číslo řádku), komparační funkce bude navracet zápornou hodnotu -1. Pokud by při minimalizaci účelové funkce byla hodnota účelové funkce u prvního prvku horší než u druhého, tj. $F(\mathbf{X}_1) > F(\mathbf{X}_2)$, komparační funkce navrátí kladnou hodnotu 1 (řádek 3). V ostatních případech, tj. když se hodnoty účelových funkcí obou prvků se rovnají, je výstupem funkce číslo (0).

Jak již bylo řečeno, jedná se o minimalizaci jedné účelové funkce, která může využívat další metody pro převod vícekriteriální optimalizace na jednoúčelovou optimalizaci např. pomocí váženého součtu.

Pro minimalizaci hodnot účelové funkce dvou prvků tedy platí:

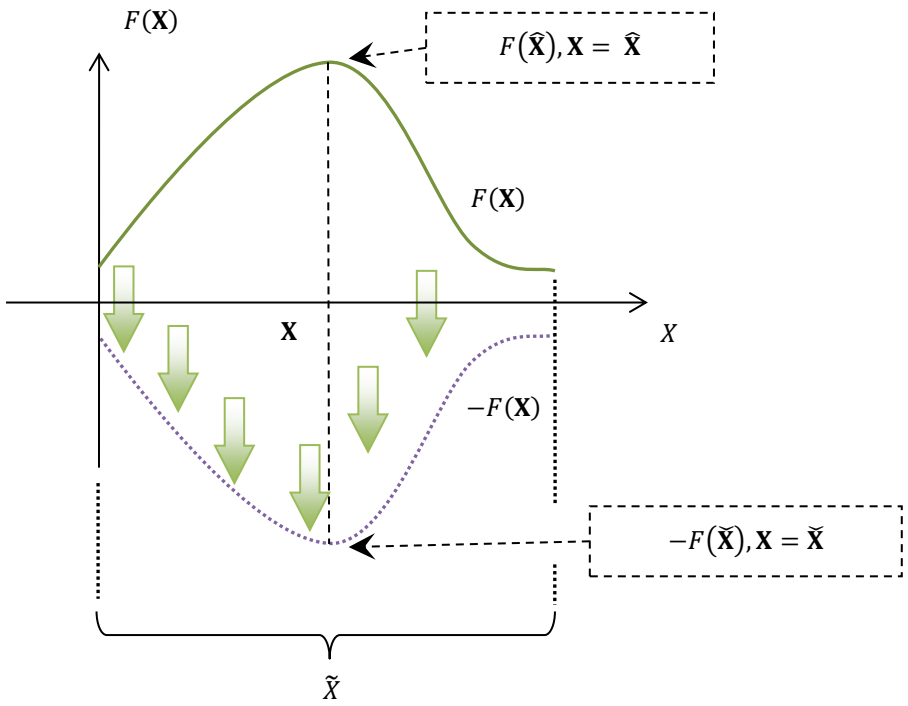
$l \longleftarrow \mathrm{CF}_{F(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2)$		
Funkce, jejímž výstupem je (-1) v případě, že první prvek je lepší než druhý – v případě minimalizace účelové funkce, hodnota účelové funkce prvního prvku je menší než hodnota účelové funkce druhého prvku. Výstup (1) nastává v případě, že první prvek je horší než druhý, a výstupem je (0) pokud jsou si prvky rovny.		
Vstup:	$\mathbf{X}_1$:	První porovnávaný prvek
Vstup:	$\mathbf{X}_2$:	Druhý porovnávaný prvek
Data:	$F(\mathbf{X})$:	Účelová funkce
Výstup:	l:	Výstup komparační funkce – možné výstupy $\{-1,1,0\}$

```
1   begin
2     if F(X1) < F(X2) then result ⟵ −1      //první prvek je lepší než druhý
3     else   if F(X1) > F(X2) then result ⟵ 1      //první prvek je horší než druhý
4            else result ⟵ 0; //prvky jsou stejné
5   end;
```

Algoritmus 3-4 Komparace hodnot účelové funkce dvou prvků v případě minimalizace účelové funkce

Pro maximalizaci hodnot účelové funkce dvou prvků platí:

$$\mathrm{CF}_{F(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2) = \begin{cases} -1 & \text{jestli } F(\mathbf{X}_1) > F(\mathbf{X}_2) \\ 1 & \text{jestli } F(\mathbf{X}_1) < F(\mathbf{X}_2) \\ 0 & \text{jinak} \end{cases} \tag{3.39}$$



Obr. 3-12 Transformace hledání maxima účelové funkce na úlohu hledání minima účelové funkce

3.7.2 Iterace

Optimalizační algoritmy zpravidla v jednotlivých iteracích generují jednotlivý prvek nebo celý seznam prvků (v kontextu např. genetických algoritmů iterací se rozumí celá populace jedinců), které jsou následně zpracovány. [15]

Základním principem iterace je tedy opakování určitého procesu v měnícím se kontextu.[6] Jinými slovy lze říci, že hlavní iterací rozumíme proběhnutí jednoho hlavního kola algoritmu, tj. opakování specifické sekvence instrukce uvnitř algoritmu. V rámci této disertační práce budeme pohlížet na iteraci v optimalizačním algoritmu, jako na proces kdy je pracováno s jedním, nebo několika prvky:

$$^{(k)}\mathbf{X}_i \ , i = \{0,1,2,\dots,m-1\}, k = \{0,1,2,\dots,\mathrm{K}-1\} \tag{3.40}$$

kde:

- k … Iterace (kolo) algoritmu.
- K … Počet iterací algoritmu – konstanta definovaná uživatelem.
- $\mathbf{X}_i$ … i-tý prvek.
- i … Index prvku (značené od nuly – index položky v seznamu).
- m … Počet vygenerovaných prvků v jedné iteraci algoritmu.

Optimalizační algoritmus obvykle na začátku vygeneruje jedno nebo více **počátečních přípustných řešení** (*Initial Possible Solution*). Druhou možností je, že počáteční přípustná řešení jsou zadána uživatelem, který může mít alespoň základní představu o průběhu účelové funkce, a tím může ovlivnit průběh hledání optima. Počáteční přípustné/á řešení musí náležet množině přípustných řešení. Generování prvku v počáteční fázi optimalizačního algoritmu je možné na základě rovnoměrného rozdělení pomocí funkce $\mathrm{Random_u}(a,b)$. Prvky je také možné generovat na základě normálního rozdělení pomocí funkce $\mathrm{Random_n}(\mu,\sigma)$.

[6] http://cs.wikipedia.org/wiki/Iterace

Prvek z geometrického pohledu na graf účelové funkce představuje bod (vektor), jehož jednotlivé souřadnice na osách (tj. hodnoty rozhodovacích proměnných) v *n*-dimensionálním prostoru jsou uloženy jako **seznam**. Přístup k jednotlivé souřadnici prvku na dané ose je realizován pomocí indexace - hranaté závorky (viz Algoritmus 3-5, řádek 5 – dále bude uváděn jen řádek).

- Create - následující algoritmus (viz Algoritmus 3-5 Generování prvku - **Create**) popisuje vytvoření prvku na základě rovnoměrného rozdělení, jehož hodnoty rozhodovacích proměnných jsou náhodná čísla generované v rozsahu dolní meze a horní meze (řádek 5). Dolní mez náhodného čísla představuje dolní mez prohledávaného intervalu dané rozhodovací proměnné. Horní mez představuje horní mez prohledávaného intervalu.

Poznámka: Algoritmus generování prvku může mít jinou formu. Ta převážně závisí na typu řešené úlohy. Obecně bychom mohli vyjádřit generování prvku jako operaci Create( ). Prvek totiž může být generován např. na základě normálního rozdělení, v případě, že uživatel má představu o průběhu účelové funkce. Vygenerováním počátečního přípustného řešení (přípustných řešení ve formě prvků) v okolí, kde se nachází slibný extrém funkce, může významně ovlivnit konvergenci algoritmu.

$\mathbf{X} \longleftarrow \text{Create}\,(A, B)$		
Funkce, jejímž výstupem je náhodně vygenerovaný prvek na základě rovnoměrného rozdělení v mezích intervalu prohledávaného prostoru.		
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	n:	Dimenze prostoru všech rozhodovacích proměnných
Data:	j:	Čítač
Výstup:	$\mathbf{X}$:	Vygenerovaný prvek - seznam rozhodovacích proměnných

```
1   begin
2     X ⟵ ( );
3     n ⟵ min(Length(A), Length(B));
      (*zjištění počtu rozhodovacích proměnných – délky seznamu A i B by měly být stejné*)
4     for j ⟵ 0 to n − 1 do
5         X ⟵ AddListItem(X, Random_u(A[j], B[j]));
      (*vygenerování hodnoty rozhodovací proměnné, která bude vložena do prvku*)
6     result ⟵ X;
7   end;
```

Algoritmus 3-5 Generování prvku - Create

Pokud prvek náleží množině nepřípustných řešení, musí být použita některá z metod práce s omezujícími podmínkami (např. opravný algoritmus), která prvek transformuje na prvek z množiny přípustných řešení.

V některých optimalizačních algoritmech se iterací rozumí populace (generace). Takovými algoritmy jsou zejména evoluční algoritmy, kde se v jedné iteraci algoritmu pracuje s několika (ve většině případů se všemi) prvky – jedinci v populaci (viz Algoritmus 3-6 Generování množiny prvků - **CreatePop**).

- CreatePop - následující algoritmus (viz Algoritmus 3-6) ilustruje způsob možného generování prvků. Prvek je generován pomocí předchozí funkce Create (viz Algoritmus 3-6, řádek 4 – dále bude uváděn jen řádek algoritmu). Zmíněná funkce je umístěna do cyklu s pevným krokem (řádek3). Prvky jsou postupně umísťovány na konec seznamu (řádek 4).

Poznámka: Algoritmus generování seznamu prvků může mít také jinou formu, která je závislá na typu řešené úlohy. Obecně bychom mohli vyjádřit generování prvku jako operaci $\mathrm{CreatePop}(m)$.

$X_{\mathrm{Pop}} \longleftarrow \mathrm{CreatePop}\ (m, A, B)$		
Funkce, jejímž výstupem jsou náhodně vygenerované prvky na základě rovnoměrného rozdělení v mezích intervalu prohledávaného prostoru.		
Vstup:	m:	Počet prvků v seznamu
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	n:	Dimenze prostoru všech rozhodovacích proměnných
Data:	i:	Čítač
Výstup:	X_{Pop}:	Vygenerovaný seznam prvků

```
1   begin
2     X_Pop ⟵ ( );
3     for i ⟵ 0 to m − 1 do
4         X_Pop ⟵ AddListItem (X_Pop, Create (A, B));
    (*postupné plnění seznamu prvky*)
5     result ⟵ X_Pop;
6   end;
```

Algoritmus 3-6 Generování množiny prvků - CreatePop

Definujme tzv. **relaci sousedství** (*Neighborhood Relation*), která umožňuje pro každé přípustné řešení - prvek $\mathbf{X}$ - pomocí množiny transformací $T(\mathbf{X})$ stanovit množinu přípustných bodů (prvků) „sousedících" s $\mathbf{X}$. Pro definici relace sousedství je důležitá podmínka dosažitelnosti, která požaduje, aby každé přípustné řešení bylo dosažitelné z libovolného jiného přípustného řešení postupnou aplikací relace sousedství.

Pro spojité úlohy můžeme sousedství definovat například jako m bodů na n-rozměrné kouli se středem v bodě $\mathbf{X} \in \mathbb{R}^{\boldsymbol{n}}$. Mohli bychom brát například průsečíky této *n*-rozměrné koule s osovým křížem posunutým do bodu $\mathbf{X}$ (tedy pro *n* = 2 celkem 4 body). Takto definovaná relace sousedství však nesplňuje podmínku dosažitelnosti (pro spojitý případ nesplňuje tuto podmínku žádná konečná relace). Řešením je do relace sousedství zahrnout všechny body ležící na *n*-rozměrné kouli se středem v bodě $\mathbf{X}$. Pro výpočet však potřebujeme konečné množství bodů v okolí aktuálně prohledávaného bodu, proto z této *n*-rozměrné koule vybereme náhodně m bodů. Tím se však tato metoda stává metodou stochastickou. [16]

Množinu transformací lze obecně popsat jako:

$$T = \{t_1(\mathbf{X}), t_2(\mathbf{X}), \ldots, t_o(\mathbf{X})\} \tag{3.41}$$

kde:

- T… Množina transformací.
- $t_1(\mathbf{X})$ … První transformace – argumentem je prvek $\mathbf{X}$.
- o … Počet transformací.

V oblasti evoluční algoritmů lze množinu transformací pojmout jako tzv. mutaci prvku (u evolučních algoritmů je prvek nazýván jedincem) $\mathbf{X}$. Jedná se vlastně o transformaci jednotlivých složek původního prvku $\mathbf{X}$ (hodnot souřadnic jednotlivých rozhodovacích proměnných lze v oblasti evolučních algoritmů pojmout jako jednotlivé geny jedince), které jsou uloženy jako nový zmutovaný prvek $\mathbf{X}_{\mathrm{Mut}}$. Proto následující algoritmus bude pojmenován jako mutace (viz Algoritmus 3-7). Je samozřejmé, že takto získaný nový prvek musí splňovat podmínku přípustnosti, tj. musí patřit do množiny přípustných řešení.

$\mathbf{X}_{\mathrm{Mut}} \longleftarrow \mathrm{Mutate}_{\mathrm{u}}(\mathbf{X}, E)$	
Funkce, jejímž výstupem je zmutovaný prvek (vektor představuje seznam hodnot rozhodovacích proměnných), který vznikl transformací rozhodovacích proměnných (v kontextu evolučních algoritmů mutací) vstupního prvku $\mathbf{X}$. Transformace jednotlivých složek vstupního prvku se provádí na základě vygenerování náhodné čísla generovaného podle rovnoměrného rozdělení v rozmezí délky intervalu dolní (včetně této meze) a horní meze (není prvkem tohoto intervalu) definovaného pro každou jednotlivou rozhodovací proměnnou (geometricky pro každou osu).	
Vstup:	$\mathbf{X}$: Původní prvek, jehož souřadnice budou transformovány – seznam, jehož prvky budou transformovány
Vstup:	E: Seznam obsahující jednotlivé délky intervalů (vzdálenost mezi dolní a horní mezí) pro jednotlivé rozhodovací proměnné užité pro generování náhodného čísla ($E[j] = \left\|\tilde{X}_j\right\| = \left\|b_j - a_j\right\|$)
Data:	j: Proměnná - čítač
Výstup:	$\mathbf{X}_{\mathrm{Mut}}$: Prvek - seznam, jehož složky jsou zmutované - transformované

```
1   begin
2     Randomize;
3     X_Mut ⟵ ( );
4     for j ⟵ 0 to (Length(X) − 1) do        //pro všechny rozhodovací proměnné
5       X_Mut ⟵ AddListItem(X_Mut, Random_u(X[j] − |E[j]|/2, X[j] + |E[j]|/2));
      (*přidání hodnoty souřadnice do vektoru X_Mut pomocí funkce pro generování náhodného čísla v
      mezích intervalu podle rovnoměrného rozdělení*)
6     result ⟵ X_Mut;
7   end;
```

Algoritmus 3-7 Transformace složek prvku na základě rovnoměrného rozdělení - Mutate_u

Příkladem transformace – mutace - je vygenerování prvního prvku v sousedství původního prvku v nultém iteračním kole algoritmu. Tento princip je ilustrován na příkladu iterací v algoritmu na dalším obrázku (viz Obr. 3-13 Příklad průběhu optimalizačního algoritmu - iterace algoritmu). V každé iteraci jsou vygenerovány čtyři prvky v sousedství nejlepšího prvku (bodu) z předchozího kola algoritmu. Sousedství v tomto případě představuje obdélník, jehož délky hran jsou uloženy v seznamu délek intervalů $E = [e_1, e_2]$. V matematickém zápisu je uveden identický způsob generování prvku v okolí pomocí algoritmu - funkce Mutate:

$$^{(0)}\mathbf{X}_1 = t_1\left(^{(0)}\mathbf{X}_0\right) \Leftrightarrow {}^{(0)}\mathbf{X}_1 \longleftarrow \mathrm{Mutate}_{\mathrm{u}}\left(^{(0)}\mathbf{X}_0, E\right) \tag{3.42}$$

kde:

- $\mathbf{X}_1$ … Prvek vygenerovaný pomocí transformace (mutace) v nulté iteraci.
- t_1 … První transformace – argumentem je prvek $\mathbf{X}_0$.
- $\mathbf{X}_0$ … Prvek, v jehož okolí je v nulté iteraci vygenerován pomocí transformace nový prvek.

Seznam obsahující jednotlivé délky intervalu má tedy délku $n = \mathrm{Length}(\mathbf{X}) = \mathrm{Length}(E)$, která představuje počet rozhodovacích proměnných. Každá jednotlivá délka uložená na j-té pozici v seznamu E je podkladem pro výpočet dolní a horní meze intervalu, v němž se bude generovat náhodné číslo (hodnota rozhodovací proměnné prvku), podle rovnoměrného rozdělení (viz Algoritmus 3-7, řádek 5).

Generování náhodného čísla představující souřadnici bodu na j-té ose (hodnotu rozhodovací proměnné) se děje na základě rovnoměrného rozdělení pomocí volání funkce Random_u. Dolní mez pro generování náhodného čísla je rovna polovině délky intervalu pro jednotlivou rozhodovací proměnnou (v grafu účelové funkce tato délka představuje délku okolí bodu, ve kterém bude generován další prvek) v seznamu E odečtené od souřadnice transformovaného prvku (řádek 5).

Horní mez je rovna součtu poloviny délky příslušného intervalu přičtené k souřadnici jednotlivé rozhodovací proměnné vybraného prvku (řádek 5). Absolutní hodnota délky intervalu v čitateli předchází možné chybě v zadávání délky do seznamu (řádek 5).

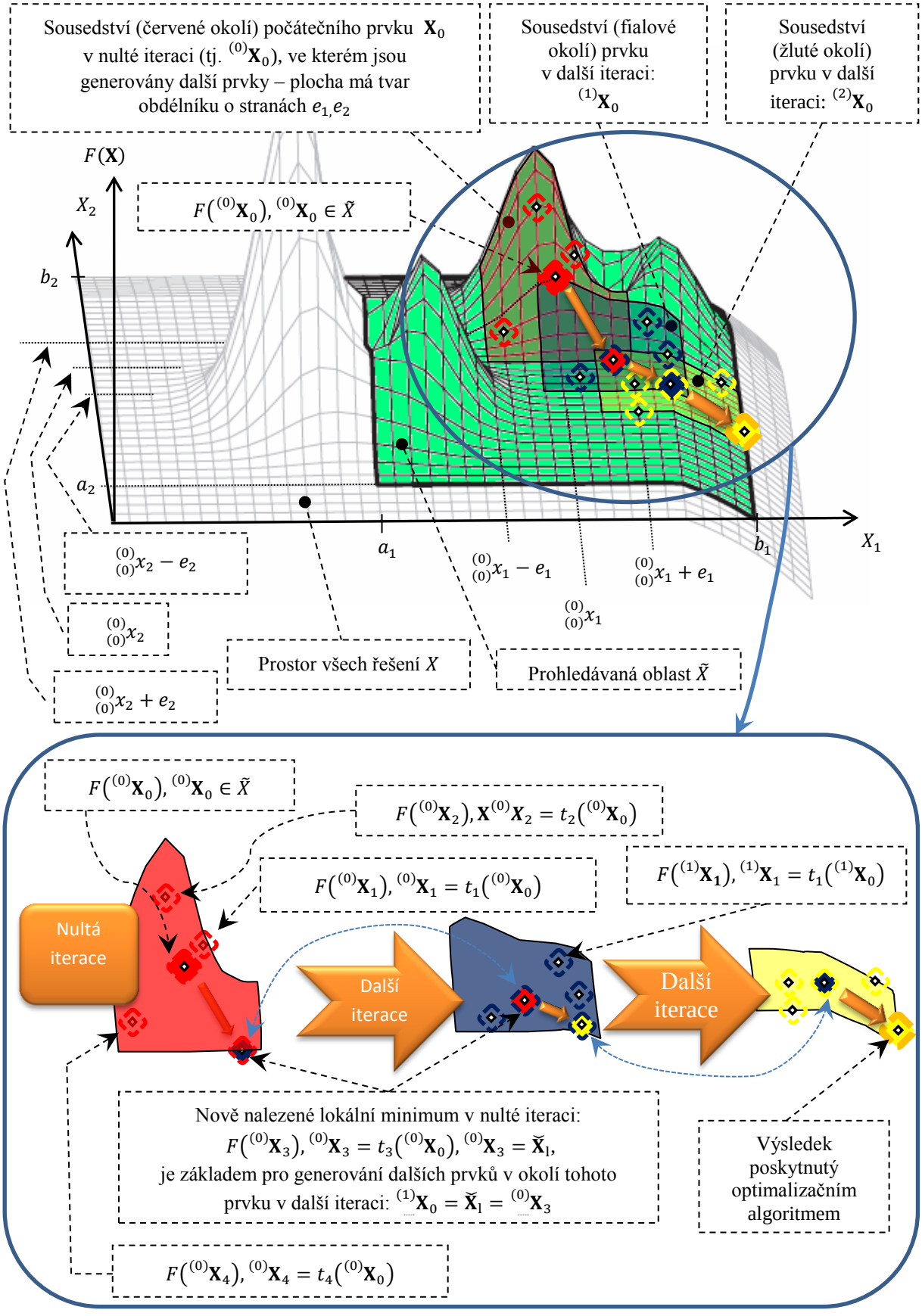
Druhou možností generování bodů v sousedství je za pomoci symetrického normálního rozdělení. Funkce generování náhodného čísla je využita při generování prvku $\mathbf{X}_{\mathrm{Mut}}$ v sousedství prvku $\mathbf{X}$ (viz Algoritmus 3-8 Transformace složek prvku na základě normálního rozdělení - $\mathbf{Mutate}_n$).

$\mathbf{X}_{\mathrm{Mut}} \longleftarrow \mathrm{Mutate}_{\mathrm{n}}(\mathbf{X}, \sigma)$	
Funkce, jejímž výstupem je zmutovaný prvek, který vznikl transformací (v kontextu evolučních algoritmů mutací) vstupního prvku $\mathbf{X}$. Transformace jednotlivých složek vstupního prvku se provádí na základě vygenerování náhodné čísla generovaného podle normálního rozdělení se směrodatnou odchylkou definovanou pro každou jednotlivou rozhodovací proměnnou (geometricky pro každou osu).	
Vstup:	$\mathbf{X}$: Původní prvek, jehož souřadnice budou transformovány – seznam, jehož prvky budou transformovány
Vstup:	σ: Seznam obsahující směrodatné odchylky pro jednotlivé rozhodovací proměnné
Data:	j: Proměnná - čítač
Výstup:	$\mathbf{X}_{\mathrm{Mut}}$: Prvek - seznam, jehož složky jsou zmutované - transformované

```
1   begin
2     Randomize;
3     X_Mut ⟵ ( );
4     for j ⟵ 0 to (Length(X) − 1) do
5         X_Mut ⟵ AddListItem(X_Mut, Random_n(X[j], σ[j]));
      (*přidání hodnoty souřadnice do vektoru X_Mut pomocí funkce pro generování náhodného čísla
      podle normálního rozdělení*)
6     result ⟵ X_Mut;
7   end;
```

Algoritmus 3-8 Transformace složek prvku na základě normálního rozdělení - $\mathbf{Mutate}_n$

Poznámka: V následujícím obrázku (viz Obr. 3-13 Příklad průběhu optimalizačního algoritmu - iterace algoritmu) jsou za účelem vysvětlení principu iterace algoritmu použity indexy prvků, které začínají hodnotou 0. To samé platí u horního indexu na levé straně prvku, který značí index iterace algoritmu.



Obr. 3-13 Příklad průběhu optimalizačního algoritmu - iterace algoritmu

3.7.3 Kritérium ukončení

Velmi důležitou složkou optimalizačního algoritmu, která zajistí, že se optimalizační algoritmus při prohledávání prostoru $\tilde{X}$ nedostane do nekonečné smyčky, je tzv. **kritérium ukončení** (*Termination Criterion*). Pokud je operace kritéria ukončení vyhodnocena logickým výrazem, optimalizační proces se zastaví a je navrácen výsledek optimalizačního procesu. Pokud je definováno více podmínek kritérií ukončení, tyto podmínky bývají většinou disjunktivní, tzn. při splnění některé z podmínek ukončení je optimalizační proces ukončen.

Příklad iterace a kritéria ukončení	
V tomto příkladu jsou jednoduchým způsobem znázorněny iterace v optimalizačním algoritmu a ukončovací kritérium	
Vstup:	$\mathrm{TerminationCriterion}()$: Typ kritéria ukončení optimalizačního algoritmu
Výstup:	k: Čítač iterací

```
1   begin
2     k ⟵ 0;    //inicializace iterace algoritmu – k-tá (nultá) populace
3     while not TerminationCriterion() do  //negace kritéria ukončení
4       …  //provádění jednotlivých operací uvnitř smyčky algoritmu
5       k ⟵ k + 1;  //zvýšení počtu iterací
6   end;
```

Algoritmus 3-9 Příklad iterace a kritéria ukončení

Ukončovacími kritérii jsou například:

1. Definování mezí např.:
   - Překročení výpočetního času, který byl nastaven. Pokud uživatel zadá hodnotu tohoto kritéria, mělo by mít nejvyšší prioritu.
   - Celkový počet iterací (vygenerovaných prvků nebo iterací algoritmu) je vyčerpán – počet vyhodnocení účelové funkce potřebný k dosažení podmínky ukončení:

$$k = \{0,1,2,\dots,K-1\} \tag{3.43}$$

   kde:

   - k … Iterace (kolo) algoritmu.
   - K … Maximální počet provedených experimentů (ohodnocení účelové funkce).

   - Pokud je řešení problému známé – **VTR** – Value To Reach (např. je známa hodnota účelové funkce). Takové kritérium ukončení bude použito zejména při testování algoritmu.

2. Optimalizační proces může být zastaven, pokud je detekováno, že už nemohou být provedeny žádné zlepšení během specifikovaného počtu iterací. To znamená, že proces konvergoval k nadějnému řešení, kde už pravděpodobně nemohou být provedena žádná další zlepšení, např.:
   - Poměr hodnoty kriteriální funkce aktuálního kandidáta řešení (nejlepšího aktuálního prvku) vůči kandidátu řešení (nejlepšímu prvku) z předchozího kola algoritmu (v případě, že jsou použity evoluční algoritmy, lze tento poměr vyjádřit např. jako poměr průměru nebo mediánu fitness populace potomků vůči průměru nebo mediánu fitness populace rodičů):

$$\omega = \begin{cases} 1 - \left(\dfrac{F\left({}^{(k)}\mathbf{X}^*\right)}{F\left({}^{(k-1)}\mathbf{X}^*\right)} \right) & \text{jestli } F(\mathbf{X}) \text{ bude minimalizována} \\ \left| 1 - \left(\dfrac{F\left({}^{(k)}\mathbf{X}^*\right)}{F\left({}^{(k-1)}\mathbf{X}^*\right)} \right) \right| & \text{jestli } F(\mathbf{X}) \text{ bude maximalizována} \end{cases} \tag{3.44}$$

$${}^{(k)}\mathbf{X}^*, k = \{0,1,2, \dots, K-1\} \tag{3.45}$$

kde:

- ω … Poměr hodnoty kriteriální funkce aktuálního kandidáta řešení vůči kandidátu řešení z předchozího kola algoritmu.
- $F\left({}^{(k)}\mathbf{X}^*\right)$ … Hodnota účelové funkce nejlepšího prvku v aktuální k-té iteraci algoritmu.
- $F\left({}^{(k-1)}\mathbf{X}^*\right)$ … Hodnota účelové funkce nejlepšího prvku v předchozí iteraci algoritmu tj. $k-1$-té iteraci algoritmu.

- Rozdíl nejlepší a nejhorší hodnoty účelové funkce, které byly doposud nalezeny - tuto podmínku ukončení je praktické formulovat tak, abychom předešli abnormálnímu ukončení programu např. v situaci, kdy prohledávání dojde do nekonečného cyklu. Proto je vhodné přidat další vstupní parametr algoritmu, což je maximální dovolený počet vyhodnocení účelové funkce (maximální počet provedených experimentů). Hledání pokračuje tak dlouho, dokud funkční hodnoty prvků v iteraci se liší více, než požadujeme nebo dokud není dosaženo nejvyššího počtu vyhodnocení účelové funkce. [11]

Z uvedených úvah je zřejmé, že lze rozlišovat následující čtyři typy ukončení prohledávání - platí při **minimalizaci cíle** (účelové funkce): [11]

- **1. typ - Korektní ukončení** - algoritmus splnil podmínku ukončení před dosažením maximálního dovoleného počtu iterací tím, že rozdíl nejlepší a nejhorší hodnoty účelové funkce jedinců v populaci, které byly doposud nalezeny, je menší než definovaná mez a současně se hodnota účelové funkce nejlepšího jedince v populaci dostatečně přiblížila hodnotě účelové funkce globálního minima.
- **2. typ - Pomalá konvergence** - algoritmus se dostatečně přiblížil hodnotou účelové funkce nejlepšího nalezeného jedince v populaci hodnotě účelové funkce globálního minima, ale prohledávání bylo ukončeno dosažením maximálního dovoleného počtu iterací.
- **3. typ - Předčasná konvergence** (*Premature Convergence*) - rozdíl hodnoty účelové funkce u nejlepšího a nejhoršího jedince v populaci, je menší než definovaná mez ale nebyl nalezen takový jedinec, který by svojí hodnotou účelové funkce byl blízký hodnotě účelové funkce globálního minima. Algoritmus skončil prohledávání v lokálním minimu nebo se jedinci populace přiblížily k sobě natolik, že jejich funkční hodnoty jsou velmi blízké.
- **4. typ - Úplné selhání** - prohledávání bylo ukončeno dosažením maximálního dovoleného počtu iterací, aniž byl nalezen jedinec blízký globálnímu minimu.

3.7.1 Množina optim

Na dalším obrázku (viz Obr. 3-14) je vidět, že pro některé případy hledání optimálního řešení je vhodné zavést pojem **množina optim**, která je množinou obsahující všechny optimální prvky:

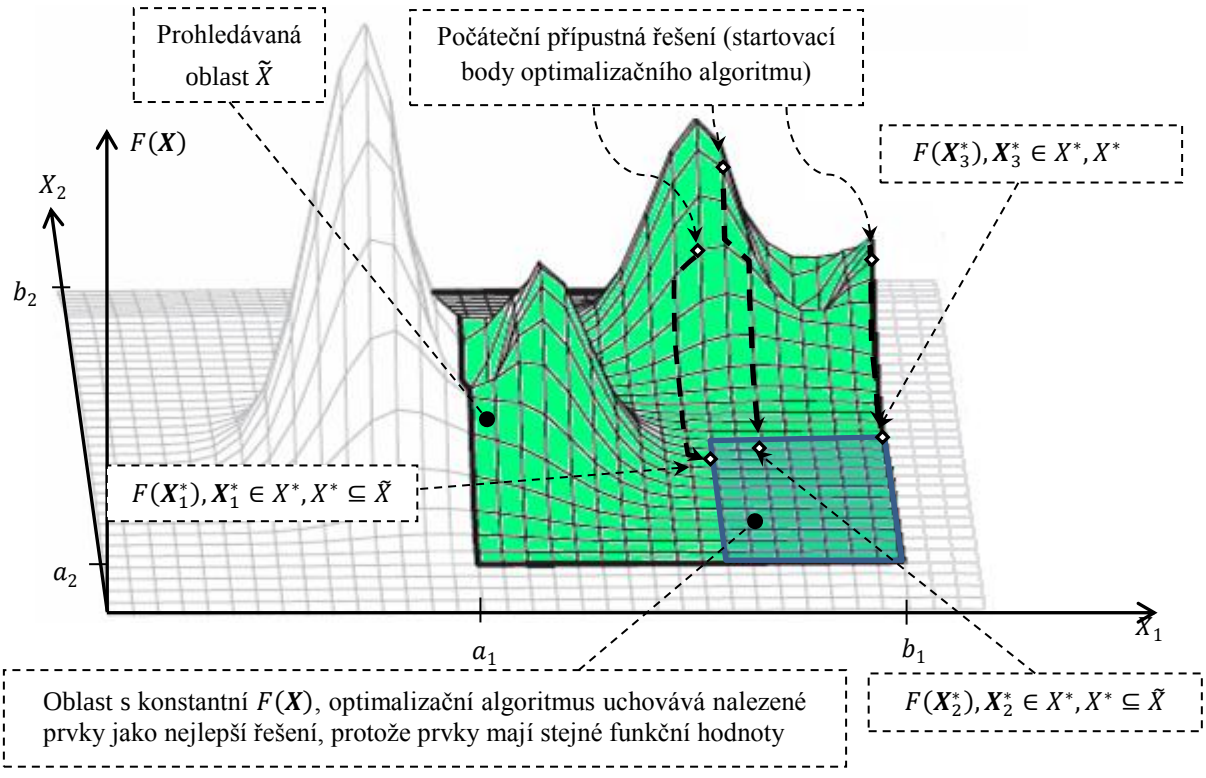
$$\mathbf{X}_i^* \in X^*, X^* \subseteq \tilde{X}, \forall i: 0 < i \leq m^* \tag{3.46}$$

kde:

- $\mathbf{X}_i^*$ … Prvek množiny optim. Nalezený možný kandidát řešení optimalizačním algoritmem.
- X^* … Množina optim.
- $\tilde{X}$ … Prohledávaný prostor optimalizačním algoritmem.
- m^* … Počet prvků množiny optim.
- i … Index prvku množiny optim.

Poznámka: V optimalizaci **kandidát řešení** (*Candidate Solution*) $\mathbf{X}_i^*$ je člen množiny možných řešení daného problému $\mathbf{X}_i^* \in X^*$. [10]

Použití množiny optim může být účelné např. v případech vícekriteriálního rozhodování, kdy namísto navrácení pouze jednoho řešení, je navrácena množina optim, uchovávající informace o tzv. cestě optimalizačního algoritmu, tj. dosud známých nejlepších kandidátů řešení (slibných prvků). Další možností je využití u případů mono-kriteriální optimalizace, kdy je předpoklad, že hodnoty účelové funkce mohou dosahovat v několika místech stejné hodnoty. V průběhu optimalizace se totiž může stát, že optimalizační algoritmus hledající např. minimum účelové funkce, mohl být několikrát spuštěn z různých počátečních řešení - prvků. Díky tomu mohl naleznout taková řešení, že jejich hodnota účelové funkce je stejná i když tyto prvky mají odlišné hodnoty souřadnic. Z následujícího obrázku je patrné, že množina optim navrácená optimalizačním algoritmem by v takovém případě obsahovala tři prvky $X^* = \{\mathbf{X}_1^*, \mathbf{X}_2^*, \mathbf{X}_3^*\}$ (viz Obr. 3-14).



Obr. 3-14 Příklad množiny optim

Dokonce i v omezeném prostoru mohou optimalizační algoritmy spoléhat pouze na informace ze vzorků v prohledávané části oblasti všech řešení – prohledávaném prostoru, protože není možné vyhodnotit účelovou funkci (funkce) pro všechny možné prvky (možné vstupy simulačních modelů). Nemůže být tudíž určeno, zda je nalezen vhodný kandidát řešení globálního optima, či nikoliv.

3.7.1.1 Aktualizace množiny optim

Jednotlivé metody a jejich stručný popis jsou převzaty z literatury - viz [10]. Tyto metody byly modifikovány pro potřeby jednotlivých optimalizačních algoritmů v rámci této disertační práce.

Jakmile je vytvořen nový prvek, stává se potenciálním kandidátem řešení. Tento prvek totiž může být lepším řešením daného problému, a proto je vhodné ho zahrnout do množiny optim. Kandidátem řešení[7] je obecně jakýkoliv prvek, který vyhovuje specifikovaným omezením. V našem případě budeme za kandidáta řešení považovat nejlepší prvek ze všech prvků vygenerovaných v jednotlivých iteracích optimalizačního algoritmu – v kontextu evolučních výpočetních technik – nejlepšího jedince v populaci.

- **UpdateOptimalSet** ($\mathrm{UpdateOptimalSet}(X^*_{\mathrm{Old}}, \mathbf{X}^*)$) funkce, která aktualizuje množinu optim, tj. přiřazuje do seznamu nový prvek - kandidáta řešení $\mathbf{X}^*$. X^*_{Old} je vlastní podmnožinou X^*_{New} obohacená o prvek $\mathbf{X}^*$. Následující algoritmus (viz Algoritmus 3-10 UpdateOptimalSet), popisuje podstatu této funkce.

$$X^*_{\mathrm{New}} = \mathrm{UpdateOptimalSet}(X^*_{\mathrm{Old}}, \mathbf{X}^*) : \mathbf{X}^* \in \tilde{X} \wedge X^*_{\mathrm{New}}, X^*_{\mathrm{Old}} \subseteq \tilde{X} \wedge X^*_{\mathrm{New}} \subseteq X^*_{\mathrm{Old}} \cup \{\mathbf{X}^*\} \tag{3.47}$$

kde:

- X^*_{New} … Nová množina optim.
- X^*_{Old} … Stará množina optim.
- $\mathbf{X}^*$ … Nový prvek, který bude testován.
- $\tilde{X}$ … Prohledávaný prostor.

$X^*_{\mathrm{New}} \longleftarrow \mathrm{UpdateOptimalSet}(X^*_{\mathrm{Old}}, \mathbf{X}^*)$	
Vytvoření nové prázdné množiny a postupné vložení optimálních prvků	
Vstup:	X^*_{Old}: Množina - seznam optim známá před vygenerováním $\mathbf{X}^*$
Vstup:	$\mathbf{X}^*$: Nový prvek, který bude testován
Výstup:	X^*_{New}: Množina - seznam optim aktualizovaná pomocí prvku $\mathbf{X}^*$

```
1   begin
2     X*New ⟵ ( );
3     for i ⟵ 0 to Length(X*Old ) − 1 do
4         if CF_F(X)(X*, X*Old[i]) > 0 then        begin
    (*nový prvek X* je horší než původní prvek X*Old[i], pak se do nové aktualizované množiny přidá
    prvek z původní množiny optim)
5             X*New ⟵ AddListItem(X*New, X*Old[i]);
6             if CF_F(X)(X*, X*Old[i]) ≥ 0 then begin
    (*nový prvek X* je stejný nebo horší než původní prvek X*Old[i], pak je jako výsledek funkce
    navrácena původní množina optim*)
7                 result ⟵ X*Old;
8                 exit;   //opuštění funkce
9             end;
```

[7] http://en.wikipedia.org/wiki/Candidate_solution

```
10        end;
11      X*New ⟵ AddListItem(X*New, X*);
        (*vložení nového prvku na konec seznamu nové aktualizované množiny optim*)
12      result ⟵ X*New;
13    end;
```

Algoritmus 3-10 UpdateOptimalSet [10]

V algoritmu je záměrně použit termín množina. Uvědomme si, že v tomto případě není důležité pořadí vkládání jednotlivých kandidátů řešení, protože po aktualizaci množiny optim novým prvkem bude následovat některá z metod ořezávání této množiny - PruneOptimalSet. Metoda zajišťuje diverzitu poskytovaných řešení – v průběhu simulačních experimentů jsou nalezena různá řešení z různých oblastí prohledávaného prostoru.

U většiny optimalizačních algoritmů je možné namísto průběžného ukládání nalezených kandidátů řešení použít tzv. extrakci množiny optimálních prvků - ExtractOptimalSet. Uplatněním takové metody na finální množinu (poslední známá množina prvků – populace - v posledním iteračním kroku, který nastal při splnění kritéria ukončení algoritmu), nám budou poskytnuta taková řešení, která postupně „přebila“ prvky z minulých kol algoritmu – při selekci byly upřednostňovány úspěšnější prvky a horší prvky byly zavrženy a nedostaly se do výběru prvků, které mohou přecházet do dalších kol algoritmu. Uplatněním takové metody ale přicházíme o jiná možná řešení problému zaznamenaných pomocí UpdateOptimalSet, které by byly průběžně zaznamenány při běhu optimalizačního algoritmu. Zejména v případě evolučních algoritmů, je v každé generaci namísto vytvoření jednoho prvku vytvořena celá množina - populace. Definujme proto k tomuto účelu operaci UpdateOptimalSetM, která bude v iteracích využívat UpdateOptimalSet (viz Algoritmus 3-10).

3.7.1.2 Extrakce optimálních prvků

Jednotlivé optimalizační metody a jejich stručný popis jsou převzaty z literatury - viz [10]. Tyto metody jsou modifikovány pro potřeby jednotlivých optimalizačních algoritmů v rámci této disertační práce.

Funkce UpdateOptimalSet pomáhá optimalizačnímu algoritmu vytvořit a spravovat seznam lepších prvků. Na konci optimalizačního procesu jsou pak poskytnuty uživateli. Nicméně, ne všechny optimalizační metody pracují s takovou množinou. Při ukončení těchto algoritmů je zapotřebí extrahovat všechny nejlepší prvky z doposud známé množiny prvků X_{Any}.

- **ExtractOptimalSet** ($\mathrm{ExtractOptimalSet}(X_{\mathrm{Any}})$) funkce, která extrahuje množinu optim X^* z dané množiny X_{Any}.

$$X^*=\mathrm{ExtractOptimalSet}(X_{\mathrm{Any}}): X^* \subseteq \tilde{X} \wedge \forall \mathbf{X}^* \in X^* \Rightarrow \exists i \in \{0, \mathrm{Length}(X_{\mathrm{Any}}) - 1\}: \mathbf{X}^* = X_{\mathrm{Any}}[i] \wedge \forall \mathbf{X}^* \in X^* \Rightarrow \nexists \mathbf{X} \in X_{\mathrm{Any}}: F(\mathbf{X}) < F(\mathbf{X}^*) \quad (3.48)$$

Následující algoritmus (viz Algoritmus 3-11) demonstruje funkci ExtractOptimalSet. Je zřejmé, že tento přístup lze použít i pro aktualizaci (viz Algoritmus 3-10 UpdateOptimalSet).

$$\mathrm{UpdateOptimalSet}(X^*_{\mathrm{Old}}, \mathbf{X}_{\mathrm{New}}) \equiv \mathrm{ExtractOptimalSet}\big(\mathrm{AddListItem}(X^*_{\mathrm{Old}}, \mathbf{X}_{\mathrm{New}})\big) \quad (3.49)$$

$X^* \longleftarrow \mathrm{ExtractOptimalSet}(X_{\mathrm{Any}})$		
Extrakce množiny optim X^* z dané množiny X_{Any}		
Vstup:	X_{Any}:	Množina, ze které jsou extrahovány prvky množiny optim X^*
Data:	i, j:	Proměnné - čítače
Výstup:	X^*:	Množina optim extrahovaná z množiny X_{Any}

```
1   begin
2     X* ⟵ X_Any;
3     i ⟵ Length(X*) − 1;
4     while i > 0 do begin
5       j ⟵ i − 1;
6       while j ≥ 0 do begin
7         if CF_F(X)(X*[i], X*[j]) < 0 then begin
      //prvek s indexem i je lepší než prvek s indexem j
8           X* ⟵ DeleteListItem(X*, j);
9           i ⟵ i − 1;
10        end
11        else
12          if CF_F(X)(X*[i], X*[j]) > 0 then begin
      //prvek s indexem i je horší než prvek s indexem j
13            X* ⟵ DeleteListItem(X*, i);
14            j ⟵ −1;
15          end;
16        j ⟵ j − 1;
17      end;
18      i ⟵ i − 1;
19    end;
20    result ⟵ X*;
21  end;
```

Algoritmus 3-11 ExtractOptimalSet

3.7.1.3 Oříznutí množiny optim

Jednotlivé optimalizační metody a jejich stručný popis jsou převzaty z literatury - viz [10]. Tyto metody jsou modifikovány pro potřeby jednotlivých optimalizačních algoritmů v rámci této disertační práce.

Při zkoumání rozlehlého prostoru jsou po proběhnutí optimalizačního algoritmu získávána nová přípustná řešení ve formě nových prvků množiny optim. Množina optim však nemůže být nekonečně velká, proto tato množina musí být ořezána takovým způsobem, abychom nepřišly o informace získané v průběhu optimalizace. Obecně lze říci:

- **PruneOptimalSet** (X^*_{New}=PruneOptimalSet(X^*_{Old})) operace, která ořezává množinu dosavadních optim X^*_{Old} získanou během optimalizace na novou množinu X^*_{New} o velikosti m^*

$$X^*_{\mathrm{New}}=\mathrm{PruneOptimalSet}(X^*_{\mathrm{Old}}): X^*_{\mathrm{New}}, X^*_{\mathrm{Old}} \subseteq \tilde{X} \quad (3.50)$$
$$\mathrm{Length}(X^*_{\mathrm{New}}) \leq m^*, m^* \in \mathbb{N} \quad (3.51)$$
$$X^*_{\mathrm{New}} \subseteq X^*_{\mathrm{Old}} \quad (3.52)$$

Ořezávání množiny optim je nejčastěji založeno na algoritmu seskupování (*Clustering Algorithm*). Princip spočívá v rozložení prohledávaného prostoru na disjunktní množiny prvků – clustery. Pro takové clustery se vypočtou jejich jádra. Pomocí nich lze např. zamítat body, které jsou dále než specifikovaná vzdálenost od jádra. Při ořezávání lze namísto algoritmu seskupování využít i jiných postupů.

4 Současný stav - globální optimalizační algoritmy

Optimalizační algoritmy lze rozdělit podle principu jejich činnosti. Následující rozdělení je jedním, ne však jediným možným rozdělením globálních optimalizačních algoritmů: [9]

- **Enumerativní** – jde o výpočet všech možných kombinací daného problému. Tento přístup je vhodný pro problémy, u nichž jsou argumenty účelové funkce (funkce, jejíž optimalizace povede k nalezení optimálních hodnot jejích argumentů) diskrétního charakteru a nabývají malého množství hodnot.
- **Deterministické** – tato skupina algoritmů je postavena pouze na rigorózních metodách klasické matematiky. Deterministické metody mají vlastnost, že v každém kroku algoritmu existuje nanejvýš jedna cesta, kterou pokračovat - determinismus. Algoritmy nevyužívají náhodných čísel při rozhodování, ale jak již bylo řečeno, při dosažení bodu se musí rozhodnout, kterou množinu kandidátů (představující možná řešení) bude prohledávat jako další.
- **Stochastické** – algoritmy tohoto typu jsou založeny na využití náhody. Jde v podstatě o čistě náhodné hledání hodnot argumentů účelové funkce s tím, že výsledkem je vždy to nejlepší řešení, jež bylo nalezeno během celého náhodného hledání. Algoritmy tohoto typu jsou obvykle:
    - Pomalé.
    - Vhodné jen pro prohledávané prostory možných řešení, jež jsou malé (malý rozsah argumentů účelové funkce).
    - Vhodné pro hrubý odhad.
- **Smíšené** – tato třída algoritmů představuje „rafinovanou“ směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou podmnožinou těchto algoritmů jsou evoluční algoritmy. Algoritmy smíšeného charakteru jsou:
    - Robustní, což znamená, že nezávisle na počátečních podmínkách velmi často naleznou kvalitní řešení, jež je reprezentováno obvykle jedním či více globálními extrémy.
    - Efektivní a výkonné, tzn., že jsou schopny nalézat kvalitní řešení během relativně malého počtu ohodnocení účelové funkce.
    - Jsou odlišné od čistě stochastických metod (díky přítomnosti podmnožiny deterministických metod).
    - Mají minimální nebo žádné požadavky na předběžné informace.
    - Jsou schopné pracovat s problémy typu „černá skříňka“, tzn., že nepotřebují ke své činnosti analytický popis problému.
    - Jsou schopny nalézt více řešení během jednoho spuštění.

Shrnou-li se tedy vlastnosti vytýčených skupin algoritmů, lze stručně konstatovat, že: [9]

- Enumerativní a stochastická optimalizace není vhodná na problémy, u nichž se prohledává rozlehlý prostor možných řešení.
- Deterministická optimalizace pracuje dobře na problémech, u nichž se pohledává úzký prostor možných řešení.
- Smíšená optimalizace je vhodná na problémy „bez omezení“ velikosti jejich prostoru možných řešení.

V zahraniční literatuře se setkáváme spíše s pojmem „**Metaheuristické** algoritmy“ namísto „Smíšených“. **Metaheuristika**[8] je heuristickou metodou pro řešení velmi obecných problémů. Metaheuristika je obecně aplikována na problémy, pro které je těžké najít nebo není praktické aplikovat specifický algoritmus nebo uplatnit heuristiku. Většina používaných metaheuristik je zaměřena na kombinatorické optimalizační problémy (u kterých se předpokládá, že by bylo těžké prozkoumávat rozlehlý prostoru řešení. Redukují proto prostor řešení na část prostoru, ve kterém je zaručeno efektivní prohledávání), nebo na problémy, které se dají přeformulovat na booleovské rovnice.

[8] http://en.wikipedia.org/wiki/Metaheuristic

4.1 Formulace podmínky optimalizační úlohy

Za účelem sjednocení popisu algoritmů budou nyní definovány podmínky pro optimalizační algoritmy, které vyhledávají globální extrém – v tomto případě **globální minimum** v souvislé oblasti - **prohledávaném prostoru**, který bude značen $\tilde{X}$, jenž je prohledávaná část prostoru všech řešení X ($\tilde{X} \subseteq X$) ve formě hranic pro každou osu - n-rozměrného kvádru (*Box Constraint*) :

$$\tilde{X} = \prod_{j=1}^{n} \tilde{X}_j = \prod_{j=1}^{n} [a_j, b_j], a_j \leq b_j \tag{4.1}$$

kde:

- $\tilde{X}_j$ … Prohledávaný interval j-té rozhodovací proměnné.
- j… Index j-té rozhodovací proměnné.
- n… Dimenze prostoru všech rozhodovacích proměnných.
- a_j … Dolní mez prohledávaného j-tého prohledávaného intervalu.
- b_j … Dolní mez prohledávaného j-tého prohledávaného intervalu.

Prvek – n-dimensionální vektor $\mathbf{X}$ je prvek prohledávané části $\tilde{X}$ a lze jej vyjádřit pomocí souřadnic v prostoru účelové funkce. Poněvadž v některých algoritmech je zapotřebí přistupovat k hodnotám souřadnic prvku (bodu - vektoru) a poněvadž záleží na pořadí jednotlivých hodnot, lze jej transformovat na seznam, kde tyto hodnoty budou indexovány podle pořadí os jednotlivých rozhodovacích proměnných:

$$\mathbf{X}[j] = x_j \forall j: j = \{0,1,2, \dots, n-1\} \tag{4.2}$$

kde:

- Význam jednotlivých parametrů je stejný jako v předpisu (4.1).

Ve většině algoritmů se prvky generují pomocí cyklů a ve většině případů prvek (v kontextu evolučních algoritmů – jedinec) bude součástí seznamu. Pro vzájemné odlišení prvků, budou prvky indexovány:

$$\mathbf{X}_i = S[i] \forall i: i = \{0,1,2, \dots, m-1\} \tag{4.3}$$

kde:

- S … Seznam prvků.
- m … Velikost (délka) seznamu S.
- i … Index (pozice) prvku v seznamu.

Pro i-tý prvek a jeho souřadnice (pro n-dimensionální prostor rozhodovacích proměnných) platí:

$$\mathbf{X}_i = [{}_{(i)}x_j] \forall {}_{(i)}x_j \in X_j: i = \{0,1,2, \dots, m-1\}, j = \{0,1,2, \dots, n-1\} \tag{4.4}$$

kde:

- $\mathbf{X}_i$ … i-tý prvek.
- i… Index prvku.
- ${}_{(i)}x_j$ … Hodnota j-té rozhodovací proměnné X_j - geometricky znázorňuje souřadnici i-tého prvku na ose X_j.
- X_j … j-tá rozhodovací proměnná.
- j… Index j-té rozhodovací proměnné – index osy.
- n… Dimenze prostoru všech rozhodovacích proměnných.

- m … Počet vygenerovaných bodů v jednom kole algoritmu – počet iterací (v kontextu evolučních algoritmů - počet vygenerovaných jedinců v populaci).

Pro hodnoty j-té rozhodovací proměnné vektoru $\mathbf{X}_i$ platí, že se budou pohybovat v rozmezí prohledávaného prostoru $\tilde{X}$:

$$\forall {}_{(i)}x_j, i = \{0,1,2,\dots,m-1\}, j = \{0,1,2,\dots,n-1\} \Rightarrow \left({}_{(i)}x_j \geq a_j\right) \wedge \left({}_{(i)}x_j \leq b_j\right) \tag{4.5}$$

kde:

- ${}_{(i)}\boldsymbol{x_j}, j, i, m, n, a_j, b_j$ … Význam jednotlivých parametrů je stejný jako v předpisu (4.1), (4.4).

Za účelem názornosti bude vymezení rozsahu souřadnic rozhodovacích proměnných prvku jediným omezením $g(\mathbf{X})$, kladeným na prvky, které jsou součástí (vlastní podmnožinou) prohledávaného prostoru $\tilde{X}$. V tomto případě tedy platí, že pokud se tento prvek nachází v prohledávaném prostoru, splňuje podmínku přípustnosti řešení:

$$\mathbf{X} \in \tilde{X} \Leftrightarrow \mathbf{X} \in X_{Feasible}\left(g(\mathbf{X})\right) \tag{4.6}$$

kde:

- $g(\mathbf{X})$ … Omezení (obvykle soustava omezení kladených na hodnoty rozhodovacích proměnných).

Pokud se stane, že nově vygenerovaný prvek bude vygenerován mimo prohledávaný prostor, patří do množiny nepřípustných řešení:

$$\mathbf{X} \notin \tilde{X} \Leftrightarrow \mathbf{X} \in X_{Non-feasible} \tag{4.7}$$

V takovém případě je u prvku uplatněna některá z metod pro práce s omezujícími podmínkami a prvek je transformován tak, aby vyhovoval podmínce (4.6).

V prohledávaném prostoru $\tilde{X}$ nejsou definována **žádná** jiná omezení rozhodovacích proměnných a nejsou specifikována ani omezení týkající se účelové funkce $F(\mathbf{X})$.

4.2 Stochastické algoritmy

Při implementaci se využívá generátorů pseudonáhodných čísel. Algoritmus typicky používá náhodné veličiny jako pomocný vstup při řízení chování algoritmu v naději, že dosáhne dobrého výkonu. Formálně lze říci, že výkonnost algoritmu bude určena náhodnou proměnnou s očekáváním dobré hodnoty. Pravděpodobnostní algoritmy jsou většinou jednoduché, avšak analýza časové složitosti (v odvětví výpočetní techniky ji lze také označit jako asymptotická složitost -operační náročnost) algoritmu je často náročná. V optimalizačních algoritmech je také náhodně generováno počáteční přípustné řešení (nebo celá množina počátečních přípustných řešení). Pokud má uživatel představu o průběhu účelové funkce, může počáteční přípustné řešení definovat sám. V mnoha případech, zvláště u gradientních metod, tím ovlivní průběh hledání globálního extrému účelové funkce.

4.2.1 Náhodné prohledávání (*Random Search*)

Náhodné prohledávání, nebo také někdy nazývané slepé prohledávání, je možné využít zejména v případech, kdy uživatel nemá žádnou znalost o průběhu účelové funkce. V takovém případě lze jednotlivé souřadnice rozhodovacích proměnných prvků generovat jako náhodná čísla na základě náhodného rozdělení

v prohledávané oblasti. Souřadnice lze generovat také podle jiných náhodných rozdělení např. pomocí normálního rozdělení.

Pokud má uživatel představu o průběhu účelové funkce, může počáteční přípustné řešení definovat sám. V mnoha případech, zvláště u gradientních metod, tím ovlivní průběh hledání globálního extrému účelové funkce.

Kromě jednoduchosti implementace algoritmu je další výhodou jeho konvergence ke globálnímu extrému (v případě minimalizace účelové funkce konvergence ke globálnímu minimu) s rostoucím počtem iterací algoritmu. [14]

$$\lim_{k\to\infty} P(\|\mathbf{X}-\mathbf{X}^*\| < e|k) = 1, e > 0 \tag{4.8}$$

kde:

- $\lim_{k\to\infty} P$ … Limita pravděpodobnosti když k jde k nekonečnu.
- k … Iterace algoritmu.
- P … Hodnota pravděpodobnosti.
- $\|\mathbf{X}-\mathbf{X}^*\|$ … Norma vektoru – vzdálenost nalezeného řešení - prvku $\mathbf{X}$ od globálního optima - extrému účelové funkce – v případě minimalizace globálního minima.
- e … Délka okolí sousedství bodu (prvku) jednotlivé rozhodovací proměnné.

Náhodné prohledávání je globální optimalizační metoda algoritmicky velmi blízká horolezeckému algoritmu. Liší se zejména v těchto oblastech: [10]

1. V horolezeckém algoritmu jsou nové prvky generovány na základě dosud nejlepšího nalezeného prvku – v sousedství tohoto prvku. V případě náhodné prohledávání tomu tak nemusí být, zejména v případě generování prvku v rozsahu celé prohledávané oblasti. V případě normálního rozdělení generování prvku v sousedství nastává již s jistou pravděpodobností.
2. Náhodné prohledávání je čistě matematický přístup zejména v případech, kdy definičním oborem prohledávané oblasti jsou reálná čísla.

4.3 Deterministické algoritmy

Poznamenejme, že deterministické algoritmy jsou většinou založeny na využívání první a druhé derivace, a proto jsou lokální. Většinou také nejsou schopny pracovat s funkcemi typu „černá skříňka“. [16]

Algoritmy tohoto charakteru obvykle vyžadují předběžné předpoklady, jež „umožní“ této metodě dávat efektivní výsledky. Tyto předpoklady jsou, že: [9]

- Problém je lineární.
- Problém je konvexní.
- Prohledávaný prostor množných řešení je malý.
- Prohledávaný prostor množných řešení je spojitý.
- Účelová funkce je pokud možno unimodální (pouze jeden extrém).
- Mezi parametry „uvnitř“ účelové funkce nejsou nelineární interakce.
- Jsou dostupné informace typu gradient apod.
- Problém je definován a analytickém tvaru.

Následující optimalizačních algoritmus Downhill Simplex lze díky svému chování zařadit do kategorie deterministických algoritmů. Mezi variantami metody Downhill Simplex existují i takové verze algoritmu, kde simplex na počátku není generován náhodně, ale počáteční prvky (body) simplexu se záměrně volí tak, aby splňovaly podmínku nekomplanarity (afinní nezávislosti). U následujícího algoritmu je implementována náhoda

v podobě generování počátečních přípustných řešení. Algoritmus se tedy stává tzv. Stochasticko-deterministickým algoritmem.

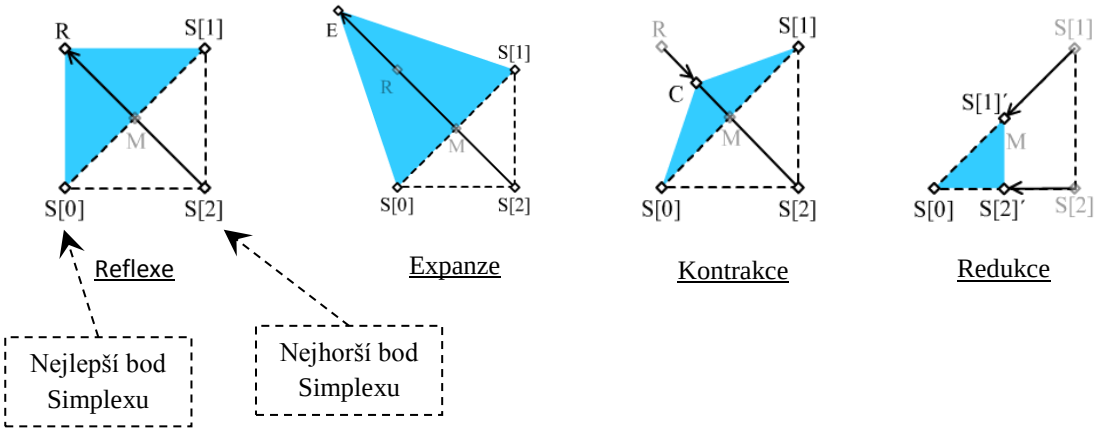
4.3.1 Downhill Simplex

Metoda Downhill Simplex (někdy též nazývaná Simplexová metoda – viz [14]) je jednoduchý a populární algoritmus pro hledání globálního extrému. Poprvé tato optimalizační metody byla prezentována Spendleyem a jeho kolektivem v roce 1962. Nealder a Mead předchozí algoritmus rozšířili o další speciální operátory, které byly navrženy pro zrychlení optimalizačního procesu. Operátory deformují simplex za účelem lepšího přizpůsobení účelové funkci.

Simplex S je množina bodů, která obsahuje minimálně $n+1$ bodů, kde n vyjadřuje rozměr prostoru rozhodovacích proměnných. Tyto body jsou nekomplanární tj. lineárně nezávislé body. Verbálně lze metodu Downhill Simplex v případě minimalizace popsat jako jakýsi „píďalkovitý“ pohyb simplexu v prohledávaném prostoru $\tilde{X}$ směrem k oblasti s nižšími funkčními hodnotami. Mezi variantami metody Downhill Simplex existují i takové, kde simplex na počátku není generován náhodně, ale počáteční body simplexu se záměrně volí tak, aby zcela jistě splňovaly podmínku nekomplanarity (afinní nezávislosti), což volně řečeno znamená, že body simplexu neleží v podprostoru s menší dimenzí, než má prohledávaný prostor $\tilde{X}$, např. při $n=2$ tři body tvořící simplex neleží na přímce. [14]

V následujícím obrázku (viz Obr. 4-1) jsou zachyceny jeho jednotlivé fáze optimalizačního algoritmu Downhill Simplex: [15]

- **Reflexe** (*Reflection*).
- **Expanze** (*Expansion*).
- **Kontrakce** (*Contraction*).
- **Redukce** (*Shrinking*).



Obr. 4-1 Princip Reflexe, Expanze, Kontrakce, Redukce

Podstata Downhill Simplex (viz [15]) bude zjednodušeně popsána pomocí jednotlivých kroků:

1. Vygenerování počáteční skupiny $n+1$ bodů simplexu (n ... počet dimenzí).
2. Výpočet těžiště (prvku **M**) ze všech prvků kromě nejhoršího prvku simplexu.
3. Vygenerování prvku **R** – Reflexe - překlopení nejhoršího (kvalita je porovnána např. podle hodnoty účelové funkce) prvku (bodu) simplexu přes těžiště simplexu **M**.
4. Jestliže prvek **R** je lepší než nejhorší prvek simplexu a zároveň horší než nejlepší prvek simplexu, je nejhorší bod simplexu nahrazen prvkem **R**.
5. Jestliže prvek **R** je lepší než nejlepší prvek simplexu, je provedena expanze – vygenerování prvku **E**.
6. Jestliže prvek **E** je lepší než prvek **R**, je nejhorší bod simplexu nahrazen prvkem **E**, jinak je nejhorší bod simplexu nahrazen prvkem **R**.

7. Jestliže prvek **R** je stejný (kvalitou) nebo horší než druhý nejhorší prvek simplexu je provedena kontrakce – vygenerování prvku **C**.
8. Jestliže prvek **C** je stejný nebo lepší než prvek **R**, je nejhorší bod simplexu nahrazen prvkem **C**.
9. Jestliže nebyla provedena reflexe, expanze nebo kontrakce, je provedena redukce – všechny prvky simplexu se posunou směrem k nejlepšímu prvku simplexu.

V některých literaturách např. [14], jsou uváděny pouze dvě základní fáze – reflexe a redukce.

Je zajímavé, že je jistá podobnost mezi evolučními algoritmy a metodou Downhill Simplex, což vede k její hybridizaci s evolučními a jinými algoritmy. V literatuře [17] se prokazuje, že metoda Downhill Simplex může být považována za evoluční algoritmus se speciálními operátory selekce a mutace. Každý prohledávací krok Nelderova a Meadova algoritmu by mohl být považován za *n*-rozměrnou reprodukci v prostoru prohledávání. Existuje také jistá podoba mezi operátorem reflexe a prohledáváním pomocí Diferenciální evoluce, PSO – optimalizace založená na rojení. Takové podobnosti jsou popsány v literatuře [18]. [15]

4.4 Gradientní metody

Standardní gradientní metody využívají určování nových řešení z předchozích a to ve směru gradientu, tj. ve směru největšího spádu v předchozím bodu (směrem k lokálnímu minimu nebo maximu).

$$\nabla F(\mathbf{X}_i)=\text{grad}\, F(\mathbf{X}_i) = \left[\frac{\partial F(\mathbf{X}_i)}{\partial_{(i)}x_j}\right] \forall_{(i)}x_j \in X_j : i = \{0,1,2,\dots,m-1\}, j = \{0,1,2,\dots,n-1\} \tag{4.9}$$

kde:

- ∇ … Operátor nabla = gradient.
- $\text{grad}\, F(\mathbf{X}_i)$ … Gradient účelové funkce prvku $\mathbf{X}_i$.
- $\frac{\partial F(\mathbf{X}_i)}{\partial_{(i)}x_j}$ … Parciální derivace účelové funkce prvku $\mathbf{X}_i$ podle hodnoty jeho *j*-té souřadnice.
- Popis dalších jednotlivých proměnných viz kapitola 4.1.

Jestliže jsme v i-tém kroku dosáhli bodu o souřadnicích $\mathbf{X}_i$ potom v dalším $(i+1)$-tém kroku bude nové řešení dáno výrazem: [19]

$$\mathbf{X}_{i+1} = \mathbf{X}_i + s \cdot \nabla F(\mathbf{X}_i) \tag{4.10}$$

kde:

- s … Míra vzdáleností mezi po sobě jdoucích kroků ($s < 0$ pro případ minimalizace $F(\mathbf{X}_i)$, $s > 0$ pro případ maximalizace $F(\mathbf{X}_i)$).

Pro použití principu gradientní metody je třeba buď spočítat uvedený gradient, nebo alespoň odhadnout směr největšího spádu účelové funkce.

4.5 Pseudo-gradientní metody

Pojem pseudo-gradientní metody je záměrně použit proto, že v optimalizačních algoritmech nelze použít stejný princip množiny transformací prvku pro výpočet všech prvků v tzv. *relaci sousedství* prvku pro určení největšího spádu (parciální derivace ve formě gradientu), jako je tomu u gradientních metod. Tento přístup byl nahrazen náhodným generováním prvků (přípustných řešení) v sousedství aktuálního prvku. Ze všech nově vygenerovaných prvků v sousedství prvku se vybere nejkvalitnější prvek (s největším rozdílem - v případě minimalizace účelové funkce, s nejmenší hodnotou účelové funkce) a tento prvek bude považován za nově nalezené řešení. Tím se optimalizační metoda stává stochasticko-gradientní metodou. V oblasti

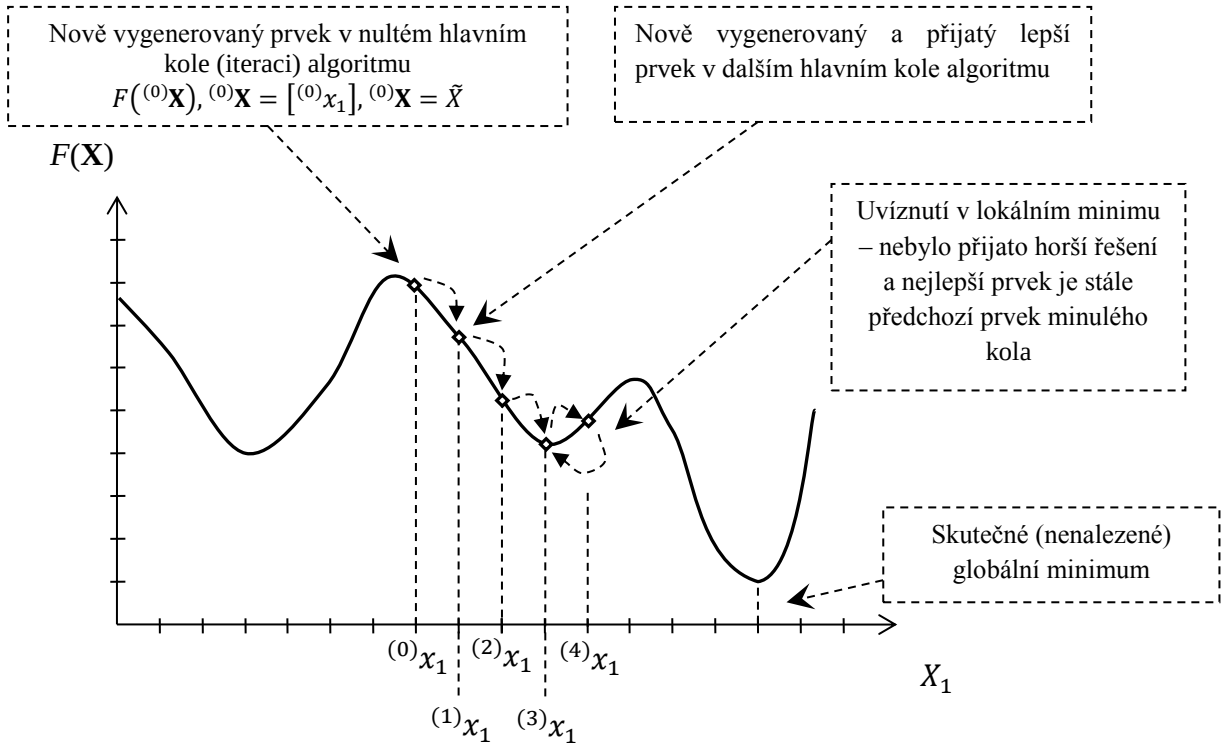
pseudogradientních metod jsou prováděny různé modifikace podstaty výběru nového řešení. Tyto modifikace jsou popsány v následujících algoritmech.

4.5.1 Stochastický horolezecký algoritmus (*Stochastic Hill Climbing*)

Metoda horolezeckého algoritmu je modifikací horolezeckého algoritmu z oblasti gradientních metod největšího spádu. Podstata stochastického horolezeckého algoritmu (viz [16]) bude zjednodušeně popsána pomocí jednotlivých kroků:

1. Na počátku optimalizačního procesu je vygenerován prvek - počáteční přípustné řešení. Tento prvek je zároveň dosud nejlepší známé řešení.
2. Pro aktuální řešení jsou náhodně vygenerovány další prvky v sousedství tohoto prvku. Tyto prvky musí splňovat podmínku přípustnosti řešení.
3. Ze všech nově vygenerovaných prvků je vybrán nejlepší prvek (v našem případě je kvalita prvku posuzována podle hodnoty účelové funkce). Tento prvek je považován za nové aktuální řešení. Po celou dobu optimalizačního procesu je průběžně ukládán dosud nejlepší nalezený prvek.
4. Během optimalizačního procesu dochází k opakování kroků 2. a 3. tak dlouho, dokud není splněno kritérium ukončení. Jako výsledek optimalizačního procesu je navrácen dosud nejlepší nalezený prvek.

Nevýhodou uvedeného optimalizačního algoritmu je, že může lehce uvíznout v lokálním extrému díky zacyklení optimalizačního algoritmu (viz Obr. 4-2 Zacyklení stochastického horolezeckého algoritmu). Tento problém lze vyřešit tak, že se algoritmus spustí několikrát z různých náhodně vygenerovaných počátečních řešení. Takový algoritmus je nazýván horolezecký algoritmus s náhodnými restarty (*Hill Climbing With Random Restarts*). Díky svému lokálnímu charakteru je optimalizační algoritmus nevhodný u optimalizace účelové funkce s více lokálními optimy.



Obr. 4-2 Zacyklení stochastického horolezeckého algoritmu

4.5.2 Stochastické zakázané prohledávání (*Stochastic Tabu Search*)

Metodu zakázaného prohledávání navrhnul koncem osmdesátých let prof. Fred Glover z Univerzity v Coloradu. Metoda je využívána i v operačním výzkumu nebo v kombinatorických úlohách. Metoda zakázaného prohledávání je modifikací předchozího stochastického horolezeckého algoritmu. U stochastického horolezeckého algoritmu může dojít k zacyklení - po určitém počtu kroků se metoda vrací k lokálnímu extrému. Metoda zakázaného prohledávání se snaží předejít uvíznutí v lokálním extrému tím, že v algoritmu je udržována krátkodobá paměť – zakázaný seznam (*Tabu List*).

Zakázaný seznam se používá ke konstrukci modifikovaného sousedství, které neobsahuje řešení vzniklá zakázanými transformacemi. Výjimku tvoří aspirační kritérium (*Aspiration Criterion*) povolující zakázanou transformaci v případě, že by vzniklé sousední řešení poskytlo lepší hodnotu účelové funkce než dosud nejlepší řešení. [13]

V našem případě bude výpočet všech bodů v okolí aktuálního řešení pomocí transformací nahrazen vygenerováním náhodných prvků v tomto okolí. Jedná se o to, že v případě, kdy jsou definičním oborem souřadnic rozhodovacích proměnných reálná čísla, by bylo časově náročné ohodnotit všechna řešení v okolí aktuálního řešení (prvku), pokud by simulační experiment trval dlouho.

Seznam zakázaných transformací bude v následujícím algoritmu obsahovat prvky, které již byly v posledních předchozích krocích vygenerovány, a tudíž nejsou akceptovány jako nové řešení.

Podstata metody zakázaného prohledávání (viz [20]) může být zjednodušeně popsána pomocí jednotlivých kroků:

1. Na počátku optimalizačního procesu je vygenerováno počáteční přípustné řešení. Prvek uložíme do zakázaného seznamu. Tento prvek je zároveň dosud nejlepší známé řešení.
2. Pro aktuální řešení jsou náhodně vygenerovány další prvky v sousedství tohoto prvku. Tyto prvky musí splňovat podmínku přípustnosti řešení. Pokud se některý z nově vygenerovaných prvků nachází v zakázaném seznamu, anebo nesplňuje výjimku v podobě aspiračního kritéria, je namísto něho generován nový prvek. Toto generování probíhá do té doby, dokud není splněna alespoň jedna z obou předchozích podmínek. Každý nový prvek je umístěn do zakázaného seznamu. Pokud došlo k překročení definované povolené délky zakázaného seznamu, jsou z tohoto seznamu odstraněny nejstarší prvky (metoda FIFO – First In First Out).
3. Ze všech nově vygenerovaných prvků je vybrán nejlepší prvek (v našem případě je kvalita prvku posuzována podle hodnoty účelové funkce). Tento prvek je považován za nové aktuální řešení. Po celou dobu optimalizačního procesu je průběžně ukládán dosud nejlepší nalezený prvek.
4. Během optimalizačního procesu dochází k opakování kroků 2. a 3. tak dlouho, dokud není splněno kritérium ukončení. Jako výsledek optimalizačního procesu je navrácen dosud nejlepší nalezený prvek.

Velikost seznamu zakázaných prvků ovlivňuje schopnost algoritmu vymanit se z lokálního optima. Pokud bude definována malá délka seznamu, může dojít k zacyklení algoritmu. Naproti tomu příliš velká délka seznamu může vést k přeskočení nadějných lokálních optim. [19]

Jedná se tedy o intenzifikaci (exploataci) a diverzifikaci (exploraci) algoritmu při vyhledávání globálního optima.

Stejně jako v případě stochastického horolezeckého algoritmu lze algoritmus spustit několikrát z náhodně vygenerovaných počátečních řešení. U této formy algoritmu lze využít k ukládání nalezených kvalitních řešení (kandidátů řešení) tzv. množinu optim. Jako výsledek bude namísto jednoho řešení poskytnuto několik řešení, ze kterých si uživatel vybere řešení jemu nejvíce vyhovující.

4.5.3 Stochastické simulované žíhání (*Stochastic Simulated Annealing*)

Počátkem 80. let Kirkpatrick, Gelatt a Vecchi ve Watsonově výzkumném centru v sídle společnosti IBM v USA a také Černý v MFF UK v Bratislavě dostali nápad, jak vytvořit algoritmus, který může řešit problémy globální optimalizace při hledání globálního optima.

V tomto algoritmu autoři využili fyzikální podstaty simulovaného žíhání, která vychází z fyzikální představy procesů při odstranění defektů krystalické mřížky. Krystal se zahřeje na vysokou teplotu a potom se pomalu ochlazuje (žíhá). Defekty krystalické mřížky mají při vysoké teplotě vysokou pravděpodobnost zániku. Pomalé ochlazování systému zabezpečuje malou pravděpodobnost vzniku nových defektů. Při žíhání se soustava snaží dostat do stavu s minimální energií – krystal bez defektů. V optimalizaci je krystal reprezentován nějakým přípustným řešením. Každému krystalu lze přiřadit energii krystalu ve formě hodnoty účelové funkce. Dalším parametrem je teplota, což je formální analogie teploty krystalu. [16]

Simulované žíhání je dalším algoritmem odvozeným od horolezeckého algoritmu. Jako u předchozí metody sekvenčního prohledávání – horolezeckého algoritmu lze užít transformace aktuálního nejlepšího prvku na nový prvek, tedy prvek v okolí. V našem případě bude znovu využito náhodného generování prvků v okolí.

Při prohledávání stavového prostoru se může snadno stát, že algoritmus uvízne v lokálním minimu. V metodě se tomu snažíme zabránit tím, že zpočátku provádíme velké změny a díky tomu se můžeme dostat z lokálního minima. Velikost změny záleží na teplotě. Čím větší je teplota, tím větší se provádí změny. V průběhu výpočtu algoritmu je teplota postupně snižována na základě rychlosti konvergence. Pokud algoritmus konverguje rychle, snižuje se teplota také rychle. Konverguje-li algoritmus pomalu, zpomalí se snižování teploty, aby se případně podařilo vyprostit z lokálního minima. [20]

Existuje mnoho variant jak snižovat teplotu. Vyjmenujeme alespoň některé z nich (specifikace jednotlivých variant je uvedena v **příloze** – kapitola 2 Stochastické simulované žíhání (*Stochastic Simulated Annealing*)) a popišme variantu simulovaného hašení, která byla využita v algoritmu:

1. Redukce teploty po definovaném počtu provedených iterací. [15]
2. Variabilní míra ochlazení. [21]
3. Variabilní míra ochlazení, která zachycuje stejný princip jako v předchozím bodu. [21]
4. Simulované hašení: [16]

$$T^{(k+1)} \leftarrow \frac{T^{(k)}}{1 + \beta * T^{(k)}} \tag{4.11}$$

kde:

- β … Nezáporná konstanta pro simulované hašení, $\beta \in \mathbb{R}^{+}, \beta < 1$.

U algoritmů simulovaného žíhání se původní prvek nahrazuje novým horším prvkem v následném procesu simulovaného žíhání s pravděpodobností dle **Metropolisova** vzorce: [22]

$$p\left(\mathbf{X}^{(k+1)} \leftarrow \mathbf{X}^{(k)}\right) \leftarrow \min\left\{1, e^{-\left(\frac{F\left(\mathbf{X}^{(k+1)}\right) - F\left(\mathbf{X}^{k}\right)}{T^{(k+1)}}\right)}\right\} \tag{4.12}$$

kde:

- p … Pravděpodobnost přijetí nového řešení.
- $\min$ … Minimum ze dvou hodnot.
- F … Účelová funkce.

Konkrétních předpisů pro výpočet této pravděpodobnosti je několik, můžeme si vybrat, který se nám pro danou úlohu lépe hodí.

Pro nastavení počáteční teploty existují různé metody. Jedna z takových metod využívá specifikace kolik procent horších prvků je uživatel ochoten akceptovat, pokud budou maximálně o danou míru v procentech hodnoty účelové funkce horší (v případě minimalizace vyšší) oproti počátečnímu řešení. Výpočet počáteční teploty zahrnující vyjmenované faktory vypadá následovně: [21]

$$T^0 = \frac{\mu}{-\ln(\phi)} * F(\mathbf{X}^0) \tag{4.13}$$

kde:

- T^0 … Počáteční teplota.
- μ … Akceptovatelná hodnota (míra) účelové funkce oproti hodnotě účelové funkce u počátečního řešení $\mu[\%]$.
- ϕ … Předpokládané procento horších prvků $\phi[\%]$.

Při vhodném nastavení počáteční teploty (dosti velká hodnota) metoda simulovaného žíhání prohledává nejprve prostor řešení silně stochasticky, přijímá i stavy s horším ohodnocením, než má současné řešení (globální charakter algoritmu). Tato vlastnost simulovaného žíhání je velmi důležitým rysem této metody, protože poskytuje možnost relativně snadno se dostat z lokálního minima a prohledat i jiné oblasti v prostoru řešení. Tato možnost se pak postupně snižuje úměrně se zmenšováním teploty. Pro velmi malé teploty připustí Metropolisovo kritérium prakticky akceptování už jen lepšího řešení než je současné (lokální charakter algoritmu). [19]

Dá se dokázat, že pokud je ochlazovací plán dostatečně pomalý a metoda generování nových přípustných řešení pokrývá celý stavový prostor, pak simulované žíhání konverguje v pravděpodobnosti ke globálnímu optimu. Protože se pohybujeme ve stavovém prostoru i směrem k horším řešením, musíme si pamatovat dosud nejlepší nalezené řešení nebo toto řešení můžeme opustit a již se k němu nevrátit. [16]

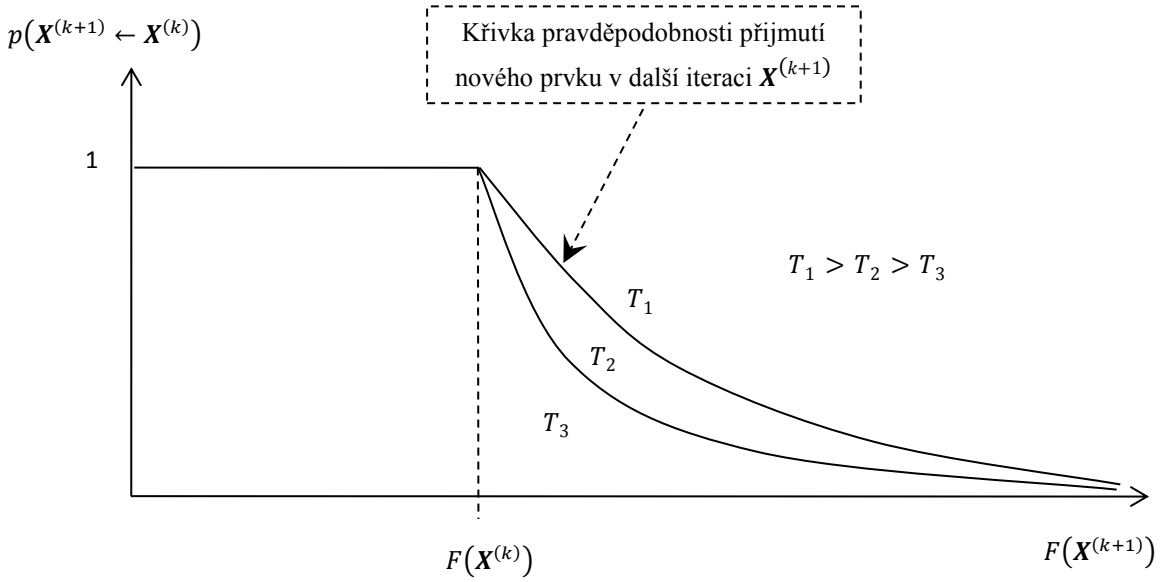
U simulovaného hašení je na začátku algoritmu pravděpodobnost přijetí horšího řešení rovna jedné. Poté co je přijat horší prvek, pravděpodobnost se snižuje spolu s klesající teplotou. To je patrné na následujícím obrázku (viz Obr. 4-3). Parametr k v obrázku vyjadřuje iteraci optimalizačního algoritmu. Parametr p vyjadřuje pravděpodobnost přijetí horšího řešení. Parametr T vyjadřuje teplotu a účelová funkce je označena jako F.

Podstata simulovaného žíhání (viz [16]) může být zjednodušeně popsána pomocí jednotlivých kroků:

1. Na počátku optimalizačního procesu je vygenerováno počáteční přípustné řešení. Tento prvek je zároveň dosud nejlepší známé řešení. Teplotu nastavíme na tzv. počáteční teplotu, která je v našem případě rovna 1.
2. Pro aktuální řešení je náhodně vygenerován prvek v sousedství. Prvek musí splňovat podmínku přípustnosti řešení.
    a. Vybereme náhodně některou složku prvku – souřadnici rozhodovací proměnné.
    b. V této složce se posuneme o náhodnou vzdálenost tak, abychom neopustili interval příslušné rozhodovací proměnné.
3. Pokud je nový prvek lepší nebo stejný než předchozí prvek (v našem případě je kvalita prvku posuzována podle hodnoty účelové funkce), nový prvek je akceptován a teplotu nesnižujeme a opakujeme krok 2. Je-li prvek horší, potom rozhodneme, zda akceptovat tento prvek:
    a. Spočítáme pravděpodobnost přijetí horšího prvku.
    b. Vygenerujeme náhodné číslo.

c. Je-li náhodné číslo větší nebo rovno, než vypočtená pravděpodobnost přijetí horší prvku, prvek neakceptujeme a teplotu nesnižujeme. Tento prvek je považován za nové aktuální řešení. Opakujeme krok 2.
d. Je-li náhodné číslo menší než vypočtená pravděpodobnost přijetí horší prvku, prvek akceptujeme a teplotu snížíme podle ochlazovacího plánu. Klesne-li teplota pod určitou definovanou mez, je teplota znovu nastavena na počáteční teplotu.
e. Po celou dobu optimalizačního procesu je průběžně ukládán dosud nejlepší nalezený prvek.

4. Během optimalizačního procesu dochází k opakování kroků 2. a 3. tak dlouho, dokud není splněno kritérium ukončení. Jako výsledek optimalizačního procesu je navrácen dosud nejlepší nalezený prvek.



Obr. 4-3 Pravděpodobnost přijmutí řešení u metody simulovaného žíhání [19]

Podle publikovaných výsledků v některých zahraničních článcích, je simulované žíhání jedním z nejúspěšnějších tradičních stochastických optimalizačních algoritmů. Proto by mohlo být rychlejší a přesnější než genetické algoritmy. Každé volání účelové funkce se využívá přímo k prohledávání stavového prostoru, nepotřebujeme žádná nadbytečná volání účelové funkce. [16]

4.5.4 Stochastické lokální prohledávání (*Stochastic Local Search*)

Metoda lokálního prohledávání je variantou gradientní metody největšího spádu, tak jako horolezecký algoritmus. Lokální prohledávání však gradient nevyužívá. Směr nejprudšího spádu se určí numericky kompletním prohledáním okolí bodu. Již sám název této optimalizační metody vypovídá o jejím nedostatku, tedy o lokálním charakteru této metody.

Nedostatek metody lokálního prohledávání (uvíznutí v lokálním extrému) lze odstranit randomizací metody, tedy že se opakovaně spustí z různého počátečního řešení a za výsledné optimum se vezme nejlepší výsledek. Stochastičnost tohoto postupu spočívá pouze v náhodném výběru počátečního řešení, ale následovně použitý optimalizační algoritmus je striktně deterministický. [23]

Metodu lze transformovat stejně jako horolezecký algoritmus. Namísto určení všech bodů v relaci sousedství pomocí množiny transformací, lze užít náhodného generování bodů v okolí prvku. Rozdíl oproti horolezeckému algoritmu je ten, že akceptujeme nejlepší prvek z nově generovaných prvků pouze tehdy, je-li vybraný prvek lepší než dosud nejlepší nalezený prvek.

Podstata stochastického lokálního prohledávání (viz [24]) bude zjednodušeně popsána pomocí jednotlivých kroků:

1. Na počátku optimalizačního procesu je vygenerován prvek - počáteční přípustné řešení. Tento prvek je zároveň dosud nejlepší známé řešení.
2. Pro aktuální řešení jsou náhodně vygenerovány další prvky v sousedství tohoto prvku. Tyto prvky musí splňovat podmínku přípustnosti řešení.
3. Ze všech nově vygenerovaných prvků je vybrán nejlepší prvek (v našem případě je kvalita prvku posuzována podle hodnoty účelové funkce). Tento prvek je považován za nové aktuální řešení, pokud je lepší než dosud nejlepší nalezený prvek.
4. Během optimalizačního procesu dochází k opakování kroků 2. a 3. tak dlouho, dokud není splněno kritérium ukončení. Jako výsledek optimalizačního procesu je navrácen poslední aktuální prvek (dosud nejlepší nalezený prvek).

4.6 Evoluční výpočetní techniky

Obecně se dá říci, že evoluční algoritmy mohou být použity na „jakoukoliv" funkci (s výjimkou typu „jehla v kupce sena" - jsou to ploché funkce, jejichž extrém je jen „dírou" v této rovině), která vrací pokud možno skalár. Žádné další přídavné informace jako gradient atd. nejsou obvykle nutné. Velmi často se evoluční algoritmy používají u funkcí, pro něž platí [9]:

1. Jsou nefraktálního typu.
2. Jsou definované na reálných, celočíselných nebo diskrétních argumentech.
3. Jsou unimodální, multimodální (jeden či více extrémů).
4. Mají různá omezení (kladená na argumenty, či hodnotu účelové funkce).
5. Jsou víceúčelové.
6. Jsou lineární, nelineární.
7. Patří do třídy NP problémů.

Přičemž pro kterýkoliv typ funkce či jejich kombinaci může navíc platit, že: [9]

a) Stupeň interakce parametrů funkce je nízký - vysoký.
b) Počet proměnných je nízký – vysoký.
c) Prostor možných řešení je:
   - Malý – velký.
   - Konečný – nekonečný.
   - Spojitý – nespojitý.

Podstatou evolučních výpočetních technik (EVT) je evoluční princip uplatňovaný v přírodě. Principem je tzv. přírodní výběr, který lze vyjádřit jako „přežití nejlépe přizpůsobeného jedince".

Poznámka: V případu evolučních algoritmů se namísto pojmu „**prvek**" využívá termín „**jedinec**" (*Individual*). Skupina jedinců pak přestavuje populaci - generaci (*Population*). Aby byla zachována autentičnost citace, v některých případech bude namísto pojmu prvek, použit termín jedinec.

Tyto techniky užívané v optimalizaci jsou inspirovány Darwinovou teorií evoluce a přirozeného výběru a Mendelovými zákony genetiky. Darwinovy poznatky můžeme shrnout do těchto následujících principů přirozeného výběru: [10]

1. Jedinci určitého druhu mají vyšší plodnost a produkují více potomků, než dospějí do fáze zralosti.
2. Při absenci externích vlivů (např. přírodní katastrofy, lidské zásahy) velikost populace druhu zůstává přibližně konstantní.

3. Pokud nenastanou žádné externí vlivy, potravinové zdroje po celou dobu zůstávají stálé, ale jsou omezené.
4. Přežití jedinců závisí na boji o limitované zdroje.
5. Žádní dva jedinci nejsou stejní.
6. Některé variace jedinců ovlivňují schopnost jejich přežití (hodnota fitness).
7. Mnoho z těchto variací jsou dědičné.
8. Jedinci s nižší hodnotou fitness mají menší pravděpodobnost reprodukce, zatímco jedinci s nejvyšší hodnotou fitness mají větší pravděpodobnost přežití a reprodukce.
9. Jedinci, kteří přežívají a reprodukují se, s velkou pravděpodobností předají jejich vlastnosti svým potomkům.
10. Druh se pomalu mění a stále se více adaptuje danému prostředí. Během tohoto procesu se může druh dokonce změnit v nový druh.

Snahou tedy je napodobit přirozený výběr, kde přežívá silnější jedinec, a tento výběr využít v oblasti optimalizace. Mírou přizpůsobení je tzv. „**fitness**“ jedince.

Jedinci s vyšší fitness mají větší pravděpodobnosti přežití a větší pravděpodobnost reprodukce svých genů do generace potomků. Kromě reprodukce se v populačním vývoji uplatňuje i tzv. mutace, což je náhodná změna genetické informace některých prvků v populaci. [14]

Nejčastějším dělením evolučních algoritmů je dělení na genetické algoritmy, genetické programování a evoluční strategie. V případě genetického programování lze říci, že se jedná o metodu strojového učení, která používá evoluční algoritmy, které postupně zlepšují populaci počítačových programů. Rozdíl genetického programování oproti genetickým algoritmům je, že u genetického programování mohou jednotlivé geny kódovat proměnné i funkce, zatímco geny u genetického algoritmu kódují pouze parametry účelové funkce.

Klasické genetické algoritmy používají operací, jako jsou **selekce**, **křížení** a **mutace** pro simulaci procesu reprodukce. Evoluční algoritmy nejsou vhodné pro aplikace, kdy lze snadno zjistit gradienty účelové funkce nebo účelová funkce je výpočtově velice náročná. Evoluční strategie obvykle vytvářejí potomky pouze modifikací (např. mutací) jednoho rodiče, tj. nemusí požívat operaci křížení.

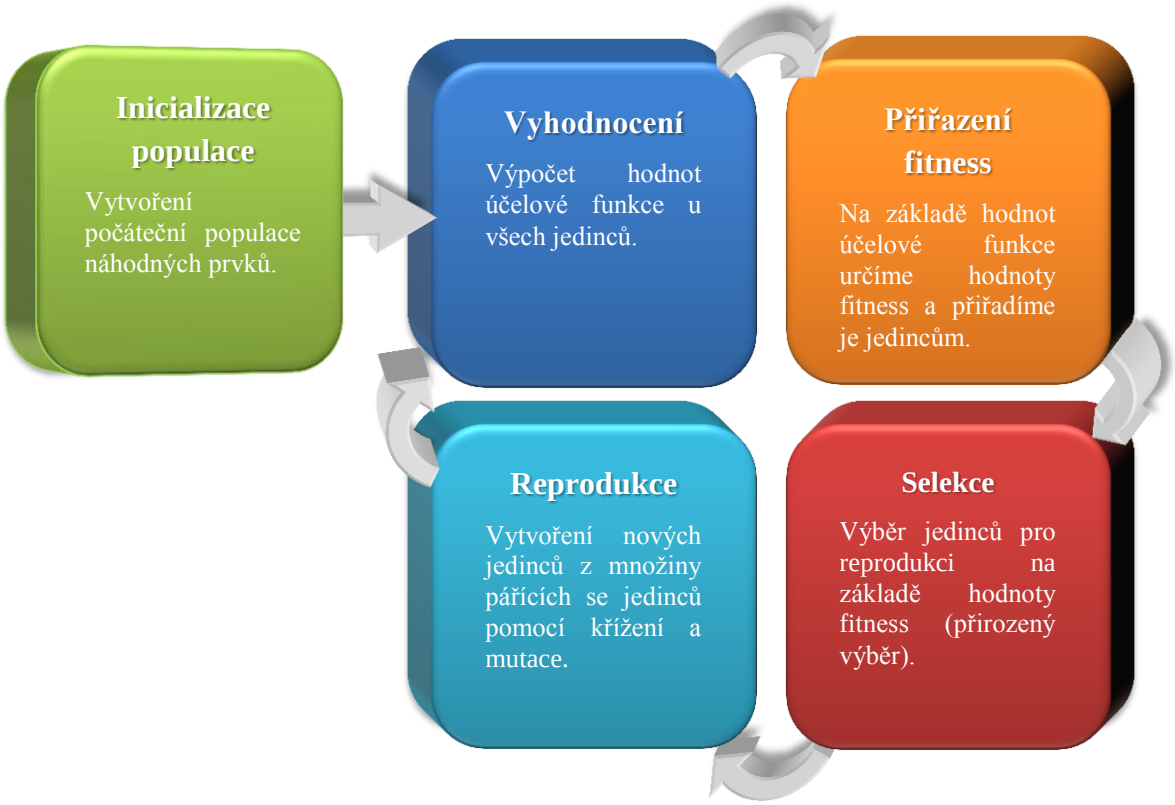
Všichni jedinci jsou v populaci implementováni pomocí téže datové struktury, ve které je zaznamenána reprezentace každého jednotlivce (v případech simulační aplikace zakódování vektorového argumentu kriteriální funkce). [19]

Na dalším obrázku (viz Obr. 4-4 Základní cyklus evolučních algoritmů) je zobrazen základní cyklus evolučních algoritmů, vyjadřující myšlenky Darwinova přirozeného výběru jedinců v populaci.

Obvykle se při inicializaci populace využívá generátoru náhodného rozdělení, pomocí kterého se vytvoří populace počátečních jedinců. Počáteční populace může být také sestavena z jedinců určených na základě znalosti úlohy – pokud máme tyto informace.

Operace vyhodnocení v sobě skrývá výpočet kvality všech jedinců v populaci. Navíc se obvykle vyhledá nejlepší jedinec a vypočtou se i jisté statistické vlastnosti populace, jako její průměr (tj. průměrné ohodnocení jedinců), rozptyl a podobně.

Selekce simuluje v EVT proces přirozeného výběru – zápasu o přežití – a vybírá jedince z předchozí populace do nové populace, obvykle na základě jejich hodnocení. Jedinci jsou v rámci selekce vybíráni zpravidla pravděpodobnostním mechanismem, přičemž lepší jedinci mají vyšší šanci být vybráni do následující generace. Již z tohoto náznaku selekčního operátoru je vidět, že průměrné ohodnocení následující populace by mělo být lepší ve srovnání s populací původní. Průměrní jedinci se totiž objeví v nové populaci přibližně ve stejném počtu jako v populaci původní, nadprůměrní v počtu vyšším, podprůměrní v počtu nižším. [10]



Obr. 4-4 Základní cyklus evolučních algoritmů [15]

Změnových či rekombinačních operátorů v EVT existuje vedle křížení a mutace více. Podstatné však je, že jsou vždy těsně svázány s konkrétní reprezentací jedinců (se strukturami) a na ní také závisí jejich skutečná implementace.

Genom (nebo též **chromozom**) můžeme představit jako prostor možných reprezentací (interpretace) jedinců. Jedinec je reprezentován pomocí **genotypu** (*Genotype*), jenž se skládá z jednotlivých **genů** (*Genes*), které vyjadřují určité vlastnosti jedince. Hodnota specifického genu se nazývá **Alela** (*Allele*). **Locus** je pozice, kde se nachází určitý gen v chromozomu (genomu). Genotyp je zobrazen do prohledávaného prostoru pomocí tzv. **fenotypu** (*Phenotype*). Základem genetických algoritmů je dobře definovat strukturu genomu. [15]

Předpokládá se, že čím déle evoluční algoritmus běží, tím lepší výsledek bude reprezentován nejlepším jedincem v populaci. Selekce může způsobit takovou homogenizaci populace, že změnové operátory nemohou již vnést do evolučního procesu nic zásadně nového. Všechny tyto jevy mohou být zahrnuty do formulace "zastavovacího pravidla". [12]

4.6.1 Populace v evolučních algoritmech

Jedním z klíčových faktorů, které od sebe odlišují různé EVT je proces výběru nové populace ze staré. Náhrada generací představuje postup (vývojovou strategii), jak z této smíšené množiny „rodičů" a „potomků" vybrat množinu, kterou budeme považovat v dalším vývoji za novou populaci. V této náhradě je zpravidla zajištěn konstantní rozsah populací. V EVT se zpravidla hovoří o těchto typech vývojových strategií:

1. **Generační** (*General Evolution*) – jde o úplnou náhradu jedné populace populací následující představující analogii životního cyklu jednoletých rostlin. [12] Strategii lze také vyjádřit jako tzv. ne-elitářskou (*Non-Elitist*), tzn., nepreferuje určité jedince – viz Algoritmus 4-1 **NonElitist** [10]
2. **Postupná** (*Steady State Evolution*) – změny podstupuje jen malá část populace a rodiče koexistují se svými potomky (model vývoje déle žijících živočich či víceletých rostlin). [12]

Tato strategie podporuje zachování lepších jedinců i v následujících populacích tzv. elitářství (*Elitist*) – zachovávají se archivy jedinců s dobrou hodnotou fitness - Algoritmus 4-2 **Elitist**

3. **Ustálená populace –** po každém křížení potomek (potomci) nahradí nejslabšího (dva nejslabší jedince). [25]
4. **Přechodové formy –** přechodové formy mezi generační a ustálenou populací. [25]

K popisu populací se u evoluční strategie v literatuře obvykle používá notace μ - značí počet rodičů a λ - počet zplozených potomků.

V případě generační vývojové strategie je tato strategie popsána jako evoluční strategie (μ, λ), kde generace potomků je tvořena μ jedinci s nejlepšími hodnotami účelové funkce z λ jedinců populace potomků. Rodičovská generace je tedy kompletně nahrazena nejlepšími jedinci z populace potomků.

V druhém případě postupné vývojové strategie, označení $(\mu + \lambda)$ je generace potomků tvořena μ jedinci s nejlepšími funkčními hodnotami ze všech $(\mu + \lambda)$ jedinců jak rodičovské populace, tak populace potomků.

V literatuře se doporučuje, aby velikost populace potomků λ byla volena několikanásobně větší než velikost rodičovské populace μ. Také se uvádí, že varianta (μ, λ) obvykle konverguje pomaleji než varianta $(\mu + \lambda)$, ale s menší tendencí ukončit prohledávání v lokálním extrému. [14]

U druhé strategie se dá mluvit o tzv. **elitářství** (*Elitism*), které zajišťuje, že alespoň jedna kopie nejlepšího jedince s dobrou hodnotou fitness (tato hodnota je většinou závislá na hodnotě účelové funkce, ale to samozřejmě závisí na použité strategii přidělování hodnoty fitness jedinci) současné generace přejde do nové generace. Použití takového přístupu je zejména vhodné, pokud losujeme v populaci jedinců, které mají téměř shodné hodnoty fitness. Znamená to, že pravděpodobnost vylosování nejlepšího jedince do nové generace bude rovna $\frac{1}{m}$, kde m bude velikost populace. Je patrné, že čím větší bude velikost populace, tím menší je pravděpodobnost vylosování nejlepšího jedince ze skupiny. Zabráníme tím ztrátě nejlepšího prvku ze skupiny.

Evoluční algoritmy využívají různých variací předchozích uvedených vývojových strategií. Následující algoritmy (viz Algoritmus 4-1 **NonElitist**, Algoritmus 4-2 **Elitist**) popisují podstatu rozdíl mezi dvěma typy vývojový strategií – neelitářské a elitářské. Popišme si nyní zjednodušeně jednotlivé bloky zachycené v algoritmech:

- $\mathrm{CreatePop}$ - tvorba počáteční náhodné populace.
- $\mathrm{TerminationCriterion}$ - formulace zastavovacího pravidla (kritéria ukončení) pro ukončení celého cyklu (algoritmu).
- UpdateOptimalSet – kontrola, zda nový jedinec bude zařazen do množiny optim. Pokud se tak stane, může to vést k tomu, že jiný jedinec z množiny optim bude muset být vyřazen.
- PruneOptimalSet – pokud je množina optim příliš velká (teoreticky může obsahovat nekonečně mnoho prvků), musí dojít k jejímu ořezání na specifikovanou velikost. Existují různé techniky ořezávání např. seskupovací algoritmy. Účelem takových technik by mělo být také zachování různorodosti jedinců.
- AssignFitness – většina z evolučních algoritmů přiřazuje hodnotu fitness (skalár) každému jedinci podle hodnoty účelové funkce vzhledem k dalším jedincům v populaci nebo množině optim. Tento přiřazovací proces je prostředkem k zachování různorodosti.
- Select – selekční algoritmus pomáhá vybírat takové jedince z populace (množiny optim), kteří se budou reprodukovat. Jedná se zde zejména o exploataci (zužitkování) - snahu zlepšovat dosavadní známé řešení pomocí malých změn, které vedou k novým podobným zlepšeným prvkům. Touto cestou lze také dosáhnout lepší výkonnosti.
- $\mathrm{ReproducePop}$ – nová populace je generována z jedinců uvnitř populace určené k páření za pomoci mutace a/nebo rekombinace.

$X^* \leftarrow \mathrm{NonElitist}(m)$		
Úplná náhrada jedné populace populací následující		
Vstup:	m:	Velikost populace X_{Pop} tj. délka seznamu $m = \mathrm{Length}(X_{\mathrm{Pop}})$

Data:	$\mathrm{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnoty fitness dvou jedinců, která je využita ve funkci UpdateOptimalSet
Data:	k: Čítač indexu generace
Data:	X_{Pop}: Populace
Data:	X_{MP}: Populace určená k páření
Výstup:	X^*: Množina nejlepších nalezených jedinců

```
1   begin
2     k ⟵ 0;
3     X_Pop ⟵ CreatePop(m);
4     while not TerminationCriterion( ) do begin
5         AssignFitness(X_Pop);
      //na základě komparační funkce CF_F(X)
6         X_MP ⟵ Select(X_Pop, m);
7         k ⟵ k + 1;
8         X_Pop ⟵ ReproducePop(X_MP, m);
9     end;
10    result ⟵ ExtractOptimalSet(X_Pop);
11  end;
```

Algoritmus 4-1 NonElitist [10]

$X^* \longleftarrow \mathrm{Elitist}(m)$	
Částečná náhrada jedné populace populací následující – koexistence potomků s rodiči	
Vstup:	m: Velikost populace X_{Pop}
Data:	$\mathrm{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnoty fitness dvou jedinců, která je využita ve funkci UpdateOptimalSet
Data:	k: Čítač indexu generace
Data:	X_{Pop}: Populace
Data:	X_{MP}: Populace určená k páření
Výstup:	X^*: Množina nejlepších nalezených jedinců

```
1   begin
2     k ⟵ 0;
3     X* ⟵ ( );
4     X_Pop ⟵ CreatePop(m);
5     while not TerminationCriterion( ) do begin
6         for each X ∈ X_Pop do X* ⟵ UpdateOptimalSet(X*, X);
7         X* ⟵ PruneOptimalSet(X*);
8         AssignFitness(X_Pop ∪ X*);
      //elitářství jedinců
9         X_MP ⟵ Select(X_Pop ∪ X*, m);
10        k ⟵ k + 1;
11        X_Pop ⟵ ReproducePop(X_MP, m);
12    end;
13    result ⟵ X*;
14  end;
```

Algoritmus 4-2 Elitist [10]

4.6.2 Reprodukce

Optimalizační algoritmy využívají shromážděné informace (do kroku *k*) pro tvorbu nových jedinců, které budou následně ohodnoceny v dalším kroku *k*+1. Existují různé metody, jak vytvořit jedince, ale v základu lze tyto metody redukovat do čtyř reprodukčních operací. Ačkoliv jsou jejich jména inspirovány genetickými algoritmy a biologickými reprodukčními mechanismy, následující popisy jsou natolik obecné, že pokrývají celou oblast globálních optimalizačních algoritmů. Je vhodné si uvědomit, že všechny následně definované operace mohou být implementovány deterministicky nebo náhodně. [10]

- Vytvoření – operace je užívána pro tvorbu nových jedinců, kteří nesouvisí s jedinci, kteří již existují. Při spuštění optimalizačního procesu, operace může být použita pro tvorbu náhodných jedinců.
- Duplikace – operace je užívána pro tvorbu přesné kopie existujícího jedince $\mathbf{X}$. Duplikace může být užitečná pro zvýšení podílu daného typu jedince v populaci v případě algoritmů, které jsou založeny na populaci, nebo v případě, že se změnila evaluační kritéria.
- Mutace - operace je užívána při tvorbě nového jedince pomocí modifikace existujícího jedince. Operace může být prováděna deterministicky nebo náhodně.
- Křížení (rekombinace) - operace je užívána při tvorbě nového jedince pomocí kombinování rysů existujících jedinců. Operace může být prováděna deterministicky nebo náhodně.

4.6.3 Přiřazení fitness

Jak již bylo řečeno, snahou většiny evolučních algoritmů a nejen jich, je napodobit přirozený výběr, kde přežívá silnější jedinec a tento výběr využít v oblasti optimalizace. Množinu prvků - jedinců lze transformovat na seznam jedinců, kde pořadí jedince v populaci je udáváno pomocí indexu v hranaté závorce, tzn. že:

$$\forall \mathbf{X} \in X_{\mathrm{Pop}} \exists i : X_{\mathrm{Pop}}[i] = \mathbf{X} \wedge \forall i \in \left[0, \mathrm{Length}(X_{\mathrm{Pop}}) - 1\right] \Rightarrow X_{\mathrm{Pop}}[i] = \mathbf{X} \tag{4.14}$$

Míru přizpůsobení popisuje skalární hodnota fitness z množiny kladných reálných čísel. Přiřazování hodnoty fitness je prováděno obecně pomocí funkce ƒ:

$$f(\mathbf{X}) \in \mathbb{R}^{+} \tag{4.15}$$

kde:

- $f(\mathbf{X})$ … Funkce, která poskytuje jako výstup hodnotu fitness jedince $\mathbf{X}$.

V následujícím textu jsou uvedeny některé možné přístupy jak vypočítat skalární hodnoty fitness, kterými jsou ohodnoceni všichni jedinci v populaci. Kromě toho, že tyto procedury mohou např. zohledňovat pořadí jedince vzhledem k ostatním jedincům, mohou také vyjadřovat hledisko, nakolik může jedinec zajistit diverzitu populace – např. pokud dochází ke zhušťování jedinců v jedné oblasti, vystavujeme se tím riziku uvíznutí v lokálním extrému.

- ❖ $\mathrm{AssignFitnessTo}$ – procedura, která přiřazuje jedincům uvnitř algoritmů vypočtené hodnoty fitness – fit:

$$\mathrm{AssignFitnessTo}\left(X_{\mathrm{Pop}}[i], fit\right) \Rightarrow \forall i \in \left[0, \mathrm{Length}(X_{\mathrm{Pop}}) - 1\right] \Rightarrow X_{\mathrm{Pop}}[i] = \mathbf{X} \exists fit \in \mathbb{R}^{+} \tag{4.16}$$

kde:

- fit … Hodnota fitness jedince, která je přiřazena i-tému jedinci z X_{Pop}.
- X_{Pop} … Seznam jedinců v populaci.
- i … Index jedince v seznamu.
- $\mathrm{Length}(X_{\mathrm{Pop}})$ … Funkce, která navrací délku seznamu X_{Pop} .
- $\mathbb{R}^{+}$… Množina kladných reálných čísel.

V následujících algoritmech je použita komparační funkce $\mathrm{CF}_{F(\mathbf{X})}$, která porovnává dva jedince seznamu (jedince v populaci) na základě hodnoty účelové funkce $F(\mathbf{X})$. Tato komparační funkce je použita záměrně, protože je schopna porovnat hodnotu i více účelových funkcí v případě víceúčelové optimalizace. Do této funkce může být totiž zakomponována některá z metod pro práci s více jednotlivými účelovými funkcemi - např. metoda váženého součtu.

Pro případ minimalizace cíle podle hodnoty jedné účelové funkce platí:

$$\mathrm{CF}_{F(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2) = \begin{cases} -1 & \text{jestli } F(\mathbf{X}_1) < F(\mathbf{X}_2) \\ 1 & \text{jestli } F(\mathbf{X}_1) > F(\mathbf{X}_2) \\ 0 & \text{jinak} \end{cases} \tag{4.17}$$

Komparace hodnot účelové funkce dvou prvků v případu minimalizace účelové funkce byla již uvedena v kapitole 3.7.1 Minimalizace (maximalizace). Případ porovnání hodnot účelové funkce dvou prvků u maximalizace účelové funkce je v podstatě stejný, s tím rozdílem, že jsou znaménka nerovnosti opačná. Následující algoritmy přiřazení fitness jsou převzaty z literatury [10]. Algoritmy byly modifikovány za účelem implementace do aplikace simulační optimalizace.

- AssignFitnessPrevalence$_1$– procedura, která přiřazuje všem jedincům (prvkům) z populace (seznamu) X_{Pop} skalární hodnotu fitness na základě komparační funkce $\mathrm{CF}_{F(\mathbf{X})}$. Každá z těchto hodnot je nepřímo úměrná počtu lepších jedinců z populace tj. $fit = \frac{1}{\text{počet lepších jedinců v populaci} + 1}$. Jedinci se stejnými hodnotami účelové funkce mají hodnotu fitness rovnu 1. Přiřazení takto vypočtené hodnoty fitness jedinci je prováděno pomocí AssignFitnessTo. Všechny uvedené algoritmy pro přiřazení fitness jsou popsány v příloze včetně pseudopascalského algoritmu (viz **příloha** - kapitola 3.1 Přiřazení fitness).
- AssignFitnessPrevalence$_2$– procedura, která přiřazuje všem jedincům z X_{Pop} skalární hodnoty fitness úměrné pořadí jedinců v populaci, které jsou lepší než právě vybraný jedinec. Jedinci se stejnými hodnotami účelové funkce mají stejnou hodnotu fitness.

Nevýhodou obou přístupů je, že neberou ohled na různorodost (diverzitu) populace.

- AssignFitnessRank - celá populace je nejprve setříděna podle komparační funkce. Jedinci je přiřazena hodnota fitness v závislosti na pořadí v setříděné populaci – prvnímu hodnota 1, dalšímu jedinci, pokud je jeho hodnota účelové funkce lepší, je přiřazeno fit o jednu vyšší než předchozímu jedinci. Jedinci se stejnými hodnotami účelové funkce mají přiřazenu stejnou hodnotu fitness.
- AssignFitnessTournament – v algoritmu je zachycen princip soutěžení jedince proti vybraným soupeřům. Jedinec soutěží v q zápasech proti r soupeřům. Obvykle bývá nastavena hodnota počtu soupeřů na $r = 1$. Vítězství jedince je určeno na základě komparační funkce, která porovnává hodnoty účelové funkce jedince proti soupeřově hodnotě účelové funkce. Soupeř je vybrán náhodně z populace X_{Pop}. Hodnota fitness je úměrná počtu vyhraných zápasů.

Poznámka: V literatuře se u optimalizačních algoritmů používá závorka ⌊ ⌋. Ta značí nejmenší celé číslo větší nebo rovné hodnotě uvnitř závorky. V případě, kdy index jedince v populaci = $\lfloor \mathrm{Random}_{\mathrm{u}}(\mathrm{Length}(X_{\mathrm{Pop}})) \rfloor$ to znamená, že indexy se pohybují pouze v oblasti od 0 až do hodnoty $\mathrm{Length}(X_{\mathrm{Pop}}) - 1$. Tento zápis se např. v programovacím jazyku Delphi nahrazuje příkazem **Trunc**.

Závorka ⌈ ⌉ značí největší celé číslo větší nebo rovné hodnotě uvnitř závorky. Synonymem této závorky v programovacím jazyku Delphi je příkaz **Round.**

4.6.4 Selekce

Selekce (*Selection*) je operace užívaná k výběru skupiny jedinců X_{MP} (*Mating Pool*[9]) pro reprodukci ze seznamu jedinců (splňují podmínku přípustnosti řešení) - populace X_{Pop}.

Selekce takových potenciální rodičů (*Parents*) určených k páření (X_{MP}) tj. k reprodukci potomků (dětí - *Children*) může probíhat buď deterministicky, nebo náhodně – záleží na řešeném problému, tj. jaká budou zvolena kritéria selekce např. hodnota fitness.

V zásadě existují dvě třídy algoritmů selekce: [15]

1. **Bez** substituce (*Without Replacement*) – každý jedinec ze seznamu kandidátů řešení X_{Pop} se může účastnit reprodukce maximálně jednou tj. výsledný seznam jedinců pro reprodukci X_{MP} obsahuje každého jedince pouze jedenkrát – není umožněna duplicita prvků. Tato třída algoritmů bude značena dolním indexem $_w$.

$$X_{\mathrm{MP}} = \mathrm{Select}_w\left(X_{\mathrm{Pop}}, m_{\mathrm{MP}}\right) \Rightarrow \forall \mathbf{X}_{\mathrm{MP}} \in X_{\mathrm{MP}} \Rightarrow \mathrm{CountItemOccurrences}(\mathbf{X}_{\mathrm{MP}}, X_{\mathrm{MP}}) = 1 \quad (4.18)$$

kde:

- m_{MP} … Požadovaná velikost populace jedinců vybraných k reprodukci - m_{MP}=$\mathrm{Length}(X_{\mathrm{MP}})$.
- X_{MP} … Populace určená k reprodukci.
- X_{Pop} … Seznam jedinců (splňujících podmínku přípustnosti řešení) ze kterých se vybírá populace X_{MP}.

V některých případech u selekce bez substituce existují jistá omezeni na m_{MP}.

2. **Se** substitucí (*With Replacement*) – skupina jedinců X_{MP}, určených k reprodukci, může obsahovat stejného jedince několikrát. To je hlavním důvodem, proč je populace jedinců určených k páření reprezentována jako seznam a ne jako množina (množina - nemůže obsahovat stejného jedince dvakrát). Tato třída algoritmů bude značena dolním indexem $_r$. Oproti předchozí variantě je varianta selekčních algoritmů se substitucí využívána více. Jedná se zejména o případ, kdy počet jedinců vybraných k reprodukci m_{MP} je větší než počet jedinců vstupního seznamu jedinců v X_{Pop}, tj. $m < m_{\mathrm{MP}}, m$=$\mathrm{Length}\left(X_{\mathrm{Pop}}\right)$.

$$X_{\mathrm{MP}} = \mathrm{Select}_r\left(X_{\mathrm{Pop}}, m_{\mathrm{MP}}\right) \Rightarrow \forall \mathbf{X}_{\mathrm{MP}} \in X_{\mathrm{MP}} \Rightarrow \mathbf{X}_{\mathrm{MP}} \in X_{\mathrm{Pop}} \wedge m_{\mathrm{MP}} = \mathrm{Length}(X_{\mathrm{MP}}) \quad (4.19)$$

Selekční algoritmy mají velký vliv na výkonnost evolučních algoritmů. Jejich chování bylo předmětem několika studií prováděných např. Goldbergem a Debem, Blicklem a Thielem, Zhong a další.

Selekce může pracovat přímo s jedinci použitím komparační funkce $\mathrm{CF}_{F(\mathbf{X})}$ nebo se vztahuje k již provedenému ohodnocení jedinců hodnoty fitness.

Pro minimalizaci hodnot fitness funkce dvou jedinců = prvků tedy platí:

$l \leftarrow \mathrm{CF}_{\mathrm{f}(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2)$		
Funkce, jejímž výstupem je $\{-1\}$ v případě, že první prvek je lepší než druhý – v případě minimalizace fitness funkce, hodnota fitness funkce prvního prvku je menší než hodnota fitness funkce druhého prvku. Výstup $\{1\}$ nastává v případě, že první prvek je horší než druhý, a výstupem je$\{0\}$, pokud jsou si prvky rovny kvalitou.		
Vstup:	$\mathbf{X}_1$:	První porovnávaný prvek
Vstup:	$\mathbf{X}_2$:	Druhý porovnávaný prvek
Data:	$\mathrm{f}(\mathbf{X})$:	Fitness funkce
Výstup:	l:	Výstup komparační funkce – možné výstupy $\{-1,1,0\}$

[9] http://en.wikipedia.org/wiki/Mating_pool

```
6   begin
7     if f(X1) < f(X2) then result ⟵ −1        //první prvek je lepší než druhý
8     else   if f(X1) > f(X2) then result ⟵ 1 //první prvek je horší než druhý
9            else result ⟵ 0; //prvky jsou stejné
10  end;
```

Algoritmus 4-3 Komparace hodnot fitness funkce dvou prvků v případě minimalizace fitness funkce

Pro maximalizaci hodnot fitness funkce dvou jedinců tedy platí:

$$\mathrm{CF}_{\mathrm{f}(\mathbf{X})}(\mathbf{X}_1,\mathbf{X}_2)=\begin{cases}-1 & \text{jestli } \mathrm{f}(\mathbf{X}_1)>\mathrm{f}(\mathbf{X}_2)\\ 1 & \text{jestli } \mathrm{f}(\mathbf{X}_1)<\mathrm{f}(\mathbf{X}_2)\\ 0 & \text{jinak}\end{cases} \tag{4.20}$$

V případě víceúčelové optimalizace algoritmy selekce pracují s algoritmy přiřazujícími jedinci skalární hodnotu fitness získanou pomocí komparační funkce (přiřazovací funkce je schopna ohodnotit jedince jednou hodnotou, i když je specifikováno více účelových funkcí).

Selekce může být použita společně s dalšími operacemi takovým způsobem, že rozšíří algoritmus, který vrací pouze jediné řešení na algoritmus vracející celý seznam řešení.

Další možnou klasifikací selekčního algoritmu je notace μ - značí počet rodičů a λ - počet zplozených potomků.

V případě evoluční strategie se selekce (μ,λ) uskutečňuje pouze mezi λ potomky, zatímco jejich rodiče jsou „zapomenuti“ bez ohledu na to jak špatné nebo dobré bylo jejich fitness vzhledem k nové generaci. Je zřejmé, že tato strategie spoléhá na přebytek narozených tj. selekce má striktní smysl pro Darwinovský přirozený výběr kdy $\lambda > \mu$. [26]

Doplňme, že velikost seznamu kandidátů řešení $\mathrm{Length}(X_{\mathrm{Pop}}) = \lambda$.

V algoritmu typu $(\mu+\lambda)$ seznam kandidátů řešení X_{Pop} obsahuje λ potomků a μ rodičů současné populace a selekce vrací μ jedinců.

V některých aplikacích je nejprve prováděna environmentální selekce[10] (druhová zděděná vlastnost bez ohledu na páření) - vliv prostředí, která redukuje počet jedinců a pak následuje selekce jedinců určených k reprodukci. [10]

4.6.4.1 Zkrácený výběr (Truncation Selection)

Zkrácený výběr, nazývaný taktéž ořízntutí, nebo deterministická selekce (*Deterministic Selectio*n, *Threshold Selection*) vrací nejlepší jedince z populace - seznamu X_{Pop}. Takový selekční algoritmus není obvykle šťastná volba, protože nezachovává diverzitu populace. Použití tohoto principu je špatné zejména v situaci, kdy by byl aplikován na množinu optim, ve které by žádný jedinec nebyl lepší než ostatní. [10], [15]

- TruncationSelect **-** funkce, která na základě hodnot fitness v seznamu X_{Pop} vybere m_{MP} jedinců a umístí je do seznamu pro reprodukci – výstup funkce. Počet jedinců by měl být menší než velikost seznamu, ze kterého bude proveden výběr do X_{MP}. Velikost populace $m = \mathrm{Length}(X_{\mathrm{Pop}})$ ze které bude proveden výběr do X_{MP}, by měla být větší než velikost seznamu jedinců určených k páření $m_{\mathrm{MP}} < m$. Běžně jsou pro m_{MP} používány hodnoty $\frac{\mathrm{Length}(X_{\mathrm{Pop}})}{2}$ nebo $\frac{\mathrm{Length}(X_{\mathrm{Pop}})}{3}$. [15] Pseudopascalský algoritmus včetně podrobnějšího popisu této metody selekce je uveden v příloze (viz **příloha** - kapitola 3.2.1 Zkrácený výběr (*Truncation Selection*)).

[10] http://en.wikipedia.org/wiki/Ecological_selection

4.6.4.2 Náhodná selekce (Random Selection)

Jak je již patrné z nadpisu, náhodná selekce vybírá jedince na základě náhodného čísla značícího index jedince v populaci, který byl vygenerován pomocí generátoru náhodných čísel s rovnoměrným rozdělením. Na tuto selekci tedy nemá vliv ani hodnota fitness jedince a ani hodnota účelové funkce jedince. Selekce se neřídí gradientem povrchu fitness, ale bude jen náhodně procházet prohledávaný prostor.

Náhodná selekce je převážně využívána ve spojení s oddělenou environmentální selekcí. Tato varianta maximálně zachovává diverzitu a je vhodnou volbou pro výběr jedinců z množiny optim.

Protože je náhodná selekce založena na rovnoměrném rozdělení, odvíjí se i chování selekční algoritmu ve stejném duchu. [15]

- $\mathrm{RandomSelect}_r$ - funkce, která na základě rovnoměrného rozdělení vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát. Pseudopascalské algoritmy včetně podrobnějšího popisu těchto metod selekcí jsou uvedeny v příloze (viz **příloha** - kapitola 3.2.2 Náhodná selekce (*Random Selection*)).
- $\mathrm{RandomSelect}_w$ - funkce, která na základě rovnoměrného rozdělení vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.

4.6.4.3 Selekce přímo úměrná fitness (Fitness Proportionate Selection)

Selekční mechanizmus imituje proces přirozeného výběru, kde má silnější jedinec větší šanci být vybrán. Potom lze očekávat, že potomstvo kvalitních jedinců se bude vyznačovat ještě lepšími vlastnostmi a tedy i vyšším ohodnocením. [13]

Takové chování lze pojmenovat jako **selekční** tlak – zvýraznění nadprůměrných jedinců a potlačení podprůměrných jedinců.

Pokud bychom použili základní algoritmus, který by využíval selekci pomocí ruletového kola - pravděpodobnost výběru je přímo úměrná ohodnocení jedince (čím lepší fitness, tím větší pravděpodobnost výběru jedince do další populace) a potlačily bychom změnové operace (křížení, mutace), zjistili bychom při pohledu na statistické parametry populace (průměrná hodnota, rozptyl kvality jedinců v posloupnosti populací) po proběhnutí několika populací, že: [12]

1. Vlivem náhodné komponenty v selekci statistické hodnoty při vývoji kolísají.
2. Rozptyl populací postupně klesá – populace se homogenizují a po čase jsou tvořeny $m_{\mathrm{MP}} = \mathrm{Length}(X_{\mathrm{MP}})$ kopiemi téhož jedince.
3. Konvergence parametrů populací není rovnoměrná v tom smyslu, že např. očekávaný nárůst střední hodnoty (vylepšování průměrné kvality populace) může být dočasně narušen.

Na straně druhé tedy selekční kritérium musí být zvoleno i tak, aby v populaci vybraných jedinců byla zachována dostatečná rozmanitost - diverzita. Jestliže toto kritérium bude favorizovat jen nejsilnější jedince, může se snadno stát, že jedinci s relativně vyšším ohodnocením velice rychle převládnou v celé populaci. Dramaticky se tím sníží rozmanitost v populaci, možnosti pro další evoluci budou silně omezené a populace může předčasně konvergovat k suboptimálnímu řešení. Je-li naopak řešení příliš volné, evoluční proces bude postupovat jen velmi pomalu a algoritmus bude často pracovat po mnoho generačních cyklů, aniž by bylo možné zaznamenávat jakýkoliv pokrok. [13]

U standardních genetických algoritmů (SGA) je úkolem selekce upřednostňovat kvalitnější jedince před horšími. Základním selekčním principem je náhodný výběr takový, že pravděpodobnost přežití každého jedince je úměrná jeho kvalitě. To je také důvod, proč je zaveden předpoklad nezápornosti účelové funkce: [12]

$$Pr(i) = c \cdot \mathfrak{f}(\mathbf{X}_i) \,, \forall i \in \left[0, \mathrm{Length}(X_{\mathrm{Pop}}) - 1\right] \tag{4.21}$$

kde:

- $Pr(i)$ … Pravděpodobnost výběru i-tého jedince do dalšího kola algoritmu.
- $\mathfrak{f}$ … Funkce, která poskytuje jako výstup hodnotu fitness jedince $\mathbf{X}_i = X_{\mathrm{Pop}}[i]$.
- c … Koeficient – určován tak, aby součet pravděpodobností přes celou populaci byl roven jedné.

Existují mnohé varianty přístupů, které využívají pravděpodobnostní rozdělení např. zbytkový stochastický výběr (*Stochastic Remainder Sampling*), universální stochastická selekce (*Stochastic Universal Selection*) atd. Nejznámější metodou je metoda Monte Carlo – ruletový mechanizmus, ruletové kolo (*Roulette Wheel Selection*). [15]

4.6.4.3.1 Selekce pomocí ruletového mechanismu

Zřejmě nejrozšířenější formou implementace přirozeného výběru je použití takzvaného ruletového mechanizmu selekce. Na rozdíl od klasické rulety, kde každé z čísel může být kvůli ekvivalentnímu obsahu předělených výsečí vybráno se stejnou pravděpodobností, v tomto případě budou favorizována kvalitnější jedinci (jedinci s vyšším ohodnocením). Takovým jedincům je na ruletovém kole přiřazena větší výseč a existuje tedy větší pravděpodobnost, že se při hře kulička zastaví právě v jejich výseči. Slabší individua mají přiřazenu menší výseč, což je sice předem nevylučuje, ale pravděpodobnost, že se kulička zastaví v jim přiřazené výseči a dojde tak k vybrání příslušného individua, je menší. Existuje několik běžně používaných způsobů, jak "upřednostňující" ruletové kolo zkonstruovat. Liší se zejména ve způsobu určování velikosti výsečí pro jednotlivá individua. Velmi rozšířený je tzv. ruletový mechanizmus selekce s pravděpodobností výběru individua přímo úměrnou jeho ohodnocení, tzv. *Fitness Proportionate Selection*, kde - jak již název napovídá - má každé individuum přiřazenou výseč, jejíž plocha je přímo úměrná ohodnocení tohoto individua.

K vytvoření takového ruletového kola stačí sečíst ohodnocení všech členů populace a potom přiřadit každému jedinci proporcionálně odpovídající kruhovou výseč.

Nechť uvažovaný seznam (populace transformovaná na seznam – záleží na pořadí jedinců) obsahuje $m \geq 0$ jedinců kde ohodnocující funkce $\mathfrak{f} \in \mathbb{R}^+$. Pravděpodobnost, s jakou bude i-tý jedinec vybrán (a odpovídající velikost kruhové výseče), je dána následujícím vztahem:

$$Pr(\mathbf{X}_i) = \frac{\mathfrak{f}(\mathbf{X}_i)}{\sum_{j=1}^{m} \mathfrak{f}(\mathbf{X}_j)} \,, \forall i \in [0, m-1] \tag{4.22}$$

kde:

- $Pr(\mathbf{X}_i)$ … Pravděpodobnost výběru i-tého jedince do dalšího kola algoritmu.
- $\mathfrak{f}(\mathbf{X}_i)$ … Funkce, která poskytuje jako výstup hodnotu fitness zkoumaného i-tého jedince $\mathbf{X}_i = X_{\mathrm{Pop}}[i]$.
- m … Velikost vstupní populace jedinců - $m = \mathrm{Length}(X_{\mathrm{Pop}})$.
- $\sum_{j=1}^{m} \mathfrak{f}(\mathbf{X}_j)$ … součet hodnot fitness všech jedinců v populaci X_{Pop}.

Vztah (4.22) je transformací vztahu pravděpodobnosti výběru i-tého jedince v populaci (4.21). Vztah (4.22) upřednostňuje jedince s vyššími hodnotami fitness - jedná se tedy o maximalizaci přiřazovací funkce $\mathfrak{f}$.

V kontextu vysvětlovaných evolučních algoritmů se jedná o minimalizaci přiřazovací funkce poskytující hodnoty fitness. Znamená to tedy, že vyšší hodnota fitness v případě minimalizace značí nevhodného jedince, zatímco menší hodnota fitness značí užitečnost jedince. Kromě toho, fitness hodnoty jsou normalizovány do rozmezí

$[0, sum]$, protože selekce přímo úměrná fitness jinak zachází s hodnotami fitness, jako jsou např. $\{0,1,2\}$ a jinak s hodnotami $\{10,11,12\}$. Whitley [27] definoval vztah, který určuje jak při výpočtu ruletového mechanismu v tomto případě postupovat: [15]

$$minf = \min\{\mathrm{f}(X_{\mathrm{Pop}}[i])\forall i \in [0, m-1]\} \quad (4.23)$$

$$maxf = \max\{\mathrm{f}(X_{\mathrm{Pop}}[i])\forall i \in [0, m-1]\} \quad (4.24)$$

$$normf(X_{\mathrm{Pop}}[i]) = \frac{maxf - \mathrm{f}(X_{\mathrm{Pop}}[i])}{maxf - minf} \quad (4.25)$$

$$Pr(X_{\mathrm{Pop}}[i]) = \frac{normf(X_{\mathrm{Pop}}[i])}{\sum_{j=1}^{m} normf(X_{\mathrm{Pop}}[j])} \quad (4.26)$$

kde:

- $minf$ … Hodnota minima funkce fitness ƒ ze všech jedinců v populaci X_{Pop}.
- $\min$ … Funkce, která poskytuje hodnotu fitness jedince z populace s nejmenší hodnotou fitness. Rozdíl oproti globálnímu minimu je, že seznam může obsahovat více stejných globálních minim.
- $maxf$ … Hodnota maxima funkce fitness ƒ ze všech jedinců v populaci X_{Pop}.
- $\max$ … Funkce, která poskytuje hodnotu fitness jedince s nejvyšší hodnotou fitness z populace. Rozdíl oproti globálnímu maximu je, že populace může obsahovat více stejných globálních maxim.
- X_{Pop} … Seznam, ve kterém je uloženo $m = \mathrm{Length}(X_{\mathrm{Pop}})$ jedinců.
- $Pr(X_{\mathrm{Pop}}[i])$ … Pravděpodobnost výběru i-tého jedince z X_{Pop} do dalšího kola algoritmu – do populace určené pro páření (reprodukci).

Návrh účelových funkcí nebo přiřazovacích funkcí fitness má významný dopad na konvergenci evolučních algoritmů. Tato selekční metoda funguje pouze tehdy, jestliže fitness jedince je úměrné míře pravděpodobnosti, že jedinec zplodí lepší potomky. Proto, ruletový mechanismus má menší efektivnost vzhledem k jiným schématům, jako jsou turnajová selekce, nebo pořadová selekce. Je zde především z důvodu kompletnosti popisu možných způsobů selekce. [15]

- ❖ RouletteWheelSelect$_r$ - funkce, která na základě ruletového mechanismu vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát. Pseudopascalské algoritmy včetně podrobnějšího popisu těchto metod selekcí jsou uvedeny v příloze (viz **příloha** - kapitola 3.2.3 Selekce přímo úměrná fitness (*Fitness Proportionate Selection*) - Selekce pomocí ruletového mechanismu (*Roulette Wheel Selection*)).
- ❖ RouletteWheelSelect$_w$ - princip ruletové selekce, ale tentokrát bez substituce, je stejný jako u předchozího algoritmu. Rozdíl je pouze v tom, že vybraní jedinci jsou odstraňovány z populace, ze které lze vybírat jedince do výsledné populace určené k páření. Zároveň musí být odstraněna z ruletového kola jejich výseč, to znamená, že musí být přepočteny horní a dolní meze výsečí všech následujících jedinců. Pokud padl výběr na prvního jedince jeho spodní mez je rovna nule.

4.6.4.4 Turnajová selekce (Tournament Selection)

Turnajová selekce byla navržena Wetzelem [28] a stala je jednou z nejoblíbenějšího a efektivního selekčního schématu ze selekčních přístupů. Vlastnosti tohoto přístupu jsou známé a byly mnohokrát analyzovány mnohými vědci např. Blickle a Thiele, Miller a Goldberg, Lee atd. [15]

Podstatou přístupu je výběr určitého počtu jedinců z populace X_{Pop} mezi nimiž se uspořádá turnaj. Vítězové zápasu se stanou jedinci selektované populace k páření X_{MP}.

Experimenty ukazují, že turnajová selekce spojená se samostatným uchováváním dosud nejlepšího řešení dává nejpříznivější výsledky. V reálných aplikacích GA se turnajová selekce používá téměř výhradně.

Poznamenejme, že pro takto upravené genetické algoritmy již neexistuje žádný aparát, který by vysvětlovat jejich chování. Přesto fungují velmi uspokojivě. Všimněme si rovněž, že všechny selekční mechanismy vycházejí jen z porovnávání kvality jedinců a že nevyžadují žádné doplňkové vlastnosti optimalizované účelové funkce, jako např. její diferencovatelnost. Jediným požadavkem je možnost vyčíslení kvality individua spolu s vytvořením rozumného selekčního tlaku. To tedy vylučuje použití GA na třídy funkcí, kterou jsou skoro všude konstantní. [12]

- TournamentSelect$_r$ - funkce, která na základě určitého počtu zápasů mezi náhodně vybranými jedinci vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát. Pseudopascalské algoritmy včetně podrobnějšího popisu těchto metod selekcí jsou uvedeny v příloze (viz **příloha** - kapitola 3.2.4 Turnajová selekce (*Tournament Selection*)).
- TournamentSelect$_{w1}$ - turnajová selekce bez substituce je, v podstatě taková varianta algoritmu, každý vítěz zápasu, už se nemusí účastnit dalších zápasů – je odstraněn z vstupní populace – listiny, kde jsou zapsáni všichni jedinci, kteří se mohou zápasit.
- TournamentSelect$_{w2}$ – podstata tohoto algoritmu je totožná s předchozím algoritmem, s tím rozdílem, že všichni jedinci – jejich indexy - jsou vybíráni do jednoho celkového kola (jedinec se ho může pouze jednou). Z tohoto seznamu je vybrán hlavní vítěz s nejmenší hodnotou fitness. Tento postup je opakován tolikrát, kolik má být vybráno jedinců k páření.

Předchozí popsané algoritmy by mohly být nazývané jako deterministická turnajová selekce, protože vítěz, který se zúčastnil každého zápasu v k soubojích vždy vstoupil do X_{MP}. V nedeterministické variantě tento případ není vždy podmínkou. Nejlepší jedinec zápasu je vybrán s pravděpodobností p, druhý jedinec s pravděpodobností $p \cdot (1-p)$, třetí jedinec s pravděpodobností $p \cdot (1-p)^2$ atd. *i*-tý nejlepší jedinec zápasu je vybrán do X_{MP} s pravděpodobností $p \cdot (1-p)^i$. [15]

- TournamentSelect$_{\mathrm{r}}^{\mathrm{p}}$ – algoritmus odráží předchozí chování turnajové selekce se substitucí. Tento algoritmus je v podstatě shodný s algoritmem TournamentSelect$_r$, pokud by byla nastavena pravděpodobnost $p = 1$.

Kromě algoritmů, které byly popsány, lze v literatuře Lee a kolektiv [29] nalézt další metody selekce založené na principu turnaje. [15]

4.6.4.5 Selekce na základě uspořádání (Ordered Selection)

Selekce na základě uspořádání je dalším přístupem, jak obejít problémy spojené se selekcí, která je úměrná hodnotě fitness. Pravděpodobnost výběru jedince je úměrná (mocninou) jeho pozici (pořadí) v setříděném seznamu jedinců populace. Implicitně parametr $k \in \mathbb{R}^+$ této selekce určuje selekční tlak. Tento tlak je roven počtu očekávaných potomků nejlepšího jedince, a proto je více podobný parametru k v turnajové selekci. Větší hodnota k zajišťuje vyšší pravděpodobnost, že jedinci, kteří nepřevládají v populaci, i když jsou ohodnoceni dobrou hodnotou účelové funkce, budou vybráni do populace určené k další reprodukci.

Podstatou metody je konverze parametru k na mocninu q pomocí níž jsou náhodná čísla vygenerované na základě rovnoměrného rozdělení zvýšeny a jsou následně použity pro indexaci seřazeného seznamu jedinců. To je dosaženo pomocí následující rovnice:

$$q = \frac{1}{1 - \frac{\log k}{\log m_{MP}}} \tag{4.27}$$

kde:

- k… Selekční tlak - počet očekávaných potomků nejlepšího jedince.

- q … Mocnina indexu náhodně vygenerovaného jedince podle rovnoměrného rozdělení.
- m_{MP} … Velikost populace určené k páření X_{MP}, musí být splněna podmínka že $k \neq m_{MP}$ protože $q = \frac{1}{1-1} = \frac{1}{0}$ … nelze dělit nulou.

❖ $\mathrm{OrderedSelect}_r$ – funkce, která na základě pravděpodobnosti výběru jedince, jenž je úměrná jeho pozici v setříděném seznamu jedinců populace podle fitness, vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát. Pseudopascalské algoritmy včetně podrobnějšího popisu těchto metod selekcí jsou uvedeny v příloze (viz **příloha** - kapitola 3.2.5 Selekce na základě uspořádání (*Ordered Selection*)).

❖ $\mathrm{OrderedSelect}_w$- selekce na základě uspořádání – bez substituce –algoritmus má shodnou podstatu s předcházejícím algoritmem se substitucí. Rozdíl je pouze v tom, že jedinec – jeho index, který byl vybrán do populace k páření, je následně odstraněn z populace X_{Pop}, ze které se vybírají jedinci do populace pro páření X_{MP}. Do populace určené k páření lze maximálně vybrat tolik jedinců, kolik je minimální hodnota z definovaného počtu jedinců nebo velikost vstupní populace.

Tyto metody jsou nezávislé na aktuálních hodnotách fitness. Turnajová selekce vytváří mnoho kopií lepší části populace a téměř žádné kopie ostatních jedinců. Selekce na základě uspořádání je zaměřena na ještě menší skupinu nejlepších jedinců, ale je také zaměřena i na horší kandidáty řešení, kteří mají pravděpodobnost přežití blížící se jedné. Jinak řečeno, zatímco turnajová selekce reprodukuje větší skupiny dobrých jedinců a zabíjí téměř většinu ostatních, selekce na základě uspořádání přiřazuje velmi vysokou plodnost velmi málo dobrým jedincům, ale také zachovává méně zdatné jedince. [15]

4.7 Evoluční strategie (*Evolution Strategy*)

Algoritmus a teorie k tomuto algoritmu je převzata z literatury [14]. Text i samotný algoritmus je modifikován pro potřeby disertační práce.

Původní nejjednodušší verzi algoritmu navrhli v šedesátých létech Schwefel a Rechenberg. Základní myšlenka je podobná slepému náhodnému prohledávání, ale rozdíl je v tom, že nový jedinec se generuje jako mutace původního jedince tak, že jednotlivé složky jedince se změní přičtením hodnot normálně rozdělených náhodných veličin: [14]

$$\mathbf{X}_{\mathrm{Mut}} = \mathbf{X} + \mathbf{U} \tag{4.28}$$

$$\mathbf{U} = \left[u_j\right], \forall j: 0 \leq j \leq n-1 \wedge u_j \sim N(0, \sigma^2\mathbf{I}) \tag{4.29}$$

kde:

- $\mathbf{X}_{\mathrm{Mut}}$ … Zmutovaný jedinec na základě normálního rozdělení.
- $\mathbf{X}$ … Původní jedinec.
- $\mathbf{U}$ … Vektor náhodný čísel.
- u_j … Náhodné číslo vygenerované na základě normálního rozdělení = prvek vektoru $\mathbf{U}$ (prvky jsou navzájem nezávislé).
- j … Index dimenze – rozhodovací proměnné $j = \{0,1,2,\ldots,n-1\}$.
- n … Počet dimenzí.
- N … Normální rozdělení s definovanou střední hodnotou 0 a rozptylem $\sigma^2\mathbf{I}$.
- m_{MP} … Velikost populace určené k páření X_{MP}.

K popisu evolučních strategií se v literatuře obvykle používá notace μ, která značí počet rodičů ($\mu = m = \mathrm{Length}(X_{\mathrm{Pop}})$) a λ, jenž značí počet zplozených potomků ($\lambda = m_{\mathrm{MP}} = \mathrm{Length}(X_{\mathrm{Arch}})$, $\lambda > \mu$).

V případě **generační** vývojové strategie je tato strategie popsána jako evoluční strategie (μ, λ), kde generace potomků je tvořena μ jedinci s nejlepšími hodnotami účelové funkce z λ jedinců populace potomků. Rodičovská generace je tedy kompletně nahrazena nejlepšími jedinci z populace potomků.

V druhém případě **postupné** vývojové strategie, označení $(\mu + \lambda)$ je generace potomků tvořena μ jedinci s nejlepšími funkčními hodnotami ze všech $(\mu + \lambda)$ jedinců jak rodičovské populace, tak populace potomků.

V literatuře se doporučuje, aby velikost populace potomků λ byla volena několikanásobně větší než velikost rodičovské populace μ. Také se uvádí, že varianta (μ, λ) obvykle konverguje pomaleji než varianta $(\mu + \lambda)$, ale s menší tendencí ukončit prohledávání v lokálním extrému. [14]

Operátorem selekce je výběr lepšího jedince z dvojice (tzv. turnajový výběr).

Rechenberg studoval vliv velikosti mutace při generování potomka, tj. hodnot σ na rychlost konvergence algoritmu a na dvou jednoduchých funkcích a odvodil tzv. pravidlo jedné pětiny úspěšnosti. Empiricky pak byla ověřena užitečnost tohoto pravidla i pro jiné funkce. Hodnoty směrodatných odchylek se v k-té iteraci optimalizačního algoritmu hledání upravují podle pravidla vyjádřeného rovnicí:

$$\sigma_j = \begin{cases} c_1 \cdot {}^{(k-1)}\sigma_j & \text{jestli } \varphi(q) < 0.2 \\ c_2 \cdot {}^{(k-1)}\sigma_j & \text{jestli } \varphi(q) > 0.2 \\ {}^{(k-1)}\sigma_j & \text{jinak} \end{cases} \tag{4.30}$$

kde:

- σ_j … Směrodatná odchylka *j*-té rozhodovací proměnné – složka v seznamu směrodatných odchylek.
- j … Index dimenze – rozhodovací proměnné.
- c_1 … Vstupní parametr, který určuje zmenšování nebo zvětšování hodnot směrodatných odchylek podle relativní četnosti úspěchu $\varphi(q)$ v předchozích q iteračních krocích algoritmu. Obvykle se volí hodnota $c_1 = 0.82$.
- c_2 … Vstupní parametr, který určuje zmenšování nebo zvětšování hodnot směrodatných odchylek podle relativní četnosti úspěchu. Obvykle se volí hodnota $c_2 = \frac{1}{c_1} = 1.22$.
- k … Iterační krok algoritmu.
- q … Počet předchozích iteračních kroků algoritmu, ve kterých se měří relativní četnost úspěchu - $\varphi(q)$. Úspěchem v případě minimalizace je, když $F(\mathbf{X}_{\mathrm{Mut}}) < F(\mathbf{X})$.
- $\mathbf{X}$ … Původní jedinec.

Autoři algoritmu evoluční strategie v minulosti museli čelit výtkám, že evoluční strategie z osvědčených evolučních operátorů evoluční strategie vůbec nevyužívá křížení. Proto byly navrženy pokročilejší varianty evoluční strategie, ve kterých je křížení obsaženo. Základní idea takového křížení spočívá v tom, že u každého jedince v populaci je kromě jeho souřadnic v prohledávaném prostoru také uchováván jeho vektor směrodatných odchylek. Vektor směrodatných odchylek potomka se pak generuje křížením s náhodně vybraným jiným jedincem. Máme-li dva rodiče s vektory směrodatných odchylek, pak vektor směrodatných odchylek jejich potomka je určován např. jako:

$$\sigma = \frac{\sigma_{\mathbf{X}_1} + \sigma_{\mathbf{X}_2}}{2} \tag{4.31}$$

kde:

- $\mathbf{X}_1$ … První rodič.
- $\mathbf{X}_2$ … Druhý rodič.
- $\sigma_{\mathbf{X}_1}$ … Vektor směrodatných odchylek prvního rodiče.

- $\sigma_{\mathbf{X}_2}$ … Vektor směrodatných odchylek druhého rodiče.
- σ … Vektor směrodatných odchylek potomka.

což je tzv. křížení průměrem nebo podle obdobného jen o trochu komplikovanějšího pravidla pro křížení.

Pro funkce, u kterých je jejich globální minimum v protáhlém údolí, jehož směr není rovnoběžný se žádnou dimenzí v prohledávaném prostoru, se užívá sofistikovanější varianta evoluční strategie, v níž u každého jedince v populaci se kromě souřadnic v prohledávaném prostoru a jeho vektoru směrodatných odchylek uchovávají i mimo diagonální prvky kovarianční matice (tzv. směrové úhly) a křížení probíhá nejen na vektorech směrodatných odchylek, ale na celé kovarianční matici. Takové varianty evoluční strategie jsou však nejen implementačně náročnější, ale také mají větší paměťové nároky i větší časovou spotřebu na jeden iterační krok. [14]

4.8 Diferenciální evoluce (*Differential Evolution*)

Algoritmus a teorie k tomuto algoritmu je převzata z literatury [14]. Text i samotný algoritmus je modifikován pro potřeby disertační práce.

Algoritmus diferenciální evoluce byl poprvé publikován v roce 1995 R. Storn a K. Price. Během několika let se stal velmi oblíbenou metodou uplatňovanou na optimalizační problém hledání globálního extrému u multimodálních funkcí. Metoda využívá tradiční evoluční operátory, jako je křížení, mutace a selekce. V algoritmu lze také využít princip adaptivity, který je uplatněn na parametr při výpočtu – mutaci - jedince (prvku – vektoru), se kterým se vybraný jedinec (rodič) bude křížit.

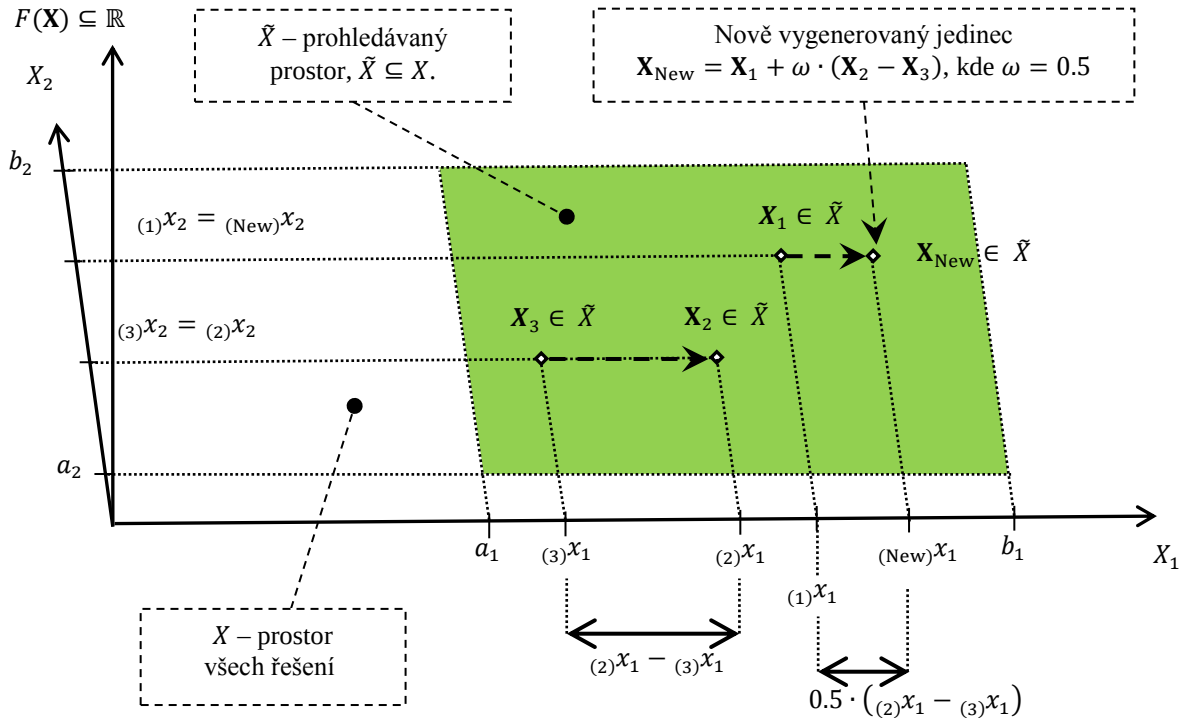
Experimentální výsledky i zkušenosti ukazují, že diferenciální evoluce konverguje rychleji než jiné stochastické algoritmy.

Princip diferenciální evoluce je založen na výběru lepšího jedince ze dvou jedinců - selekci. Těmito jedinci jsou – rodič a jeho potomek, který vznikl křížením rodiče s nově vytvořeným jedincem vzniklým mutací jedinců. Lepší jedinec z těchto dvou porovnávaných jedinců je následně zařazen do populace, která kompletně nahradí populaci rodičů. Otázkou je, jak vytvořit jedince, který bude následně křížen s rodičem. Existuje několik způsobů tvorby takového jedince. Prvním způsobem označovaným jako **RAND** je generování jedince (mutace) za pomocí tří navzájem různých náhodně vybraných jedinců z populace rodičů. Tito jedinci musí dále splňovat podmínku, že musí být také různí od právě vybraného jedince z populace rodičů. [14]

$$\mathbf{X}_{\mathrm{New}} = \mathbf{X}_1 + \omega \cdot (\mathbf{X}_2 - \mathbf{X}_3) \tag{4.32}$$

kde:

- $\mathbf{X}_{\mathrm{New}}$ … Nově vytvořený - zmutovaný jedinec.
- $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$ … První, druhý, třetí náhodně vybraný jedinec ze staré populace - populace rodičů.
- ω … Parametr adaptivního pravidla, $\omega > 0$.



Obr. 4-5 Příklad generování nového jedince pomocí postupu RAND

Druhým základním způsobem generování nového jedince pomocí mutace je metoda **BEST**. Tato metoda, na rozdíl od předchozí metody, využívá při generování nového jedince čtyři náhodně navzájem různé body a také nejlepšího jedince v populaci, který je různý od těchto jedinců. Dále platí, že nejlepší jedinec a vybraní čtyři jedinci musí být různí od právě vybraného jedince z populace rodičů. Rovnice, pomocí níž je vygenerován nový jedinec, má tvar: [14]

$$\mathbf{X}_{\mathrm{New}} = \mathbf{X}_{\mathrm{Best}} + \omega \cdot (\mathbf{X}_1 + \mathbf{X}_2 - \mathbf{X}_3 - \mathbf{X}_4) \tag{4.33}$$

kde:

- $\mathbf{X}_{\mathrm{New}}$ … Nově vytvořený - zmutovaný jedinec.
- $\mathbf{X}_{\mathrm{Best}}$ … Nejlepší jedinec v populaci – nejmenší hodnota účelové funkce při minimalizaci cíle.
- $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4$ … První, druhý, třetí, čtvrtý náhodně vybraný jedinec ze staré populace - populace rodičů.
- ω … Parametr adaptivního pravidla, $\omega > 0$.

Jak již bylo řečeno, nový potomek vzniká křížením nově vygenerovaného (vypočteného - zmutovaného) jedince a původního jedince (vybraného rodiče z populace rodičů). Takto vzniklý potomek je následně testován, zda je lepší nebo horší, než vybraný rodič. Potomek při křížení vznikne tak, že potomek je kopií vybraného rodiče. Pokud je náhodné číslo R_j menší nebo rovno specifikované konstantě *CR*, pak souřadnice rozhodovací proměnné potomka x_j je rovna souřadnici rozhodovací proměnné zmutovaného jedince $_{(\mathrm{New})}x_j$. Pokud náhodné číslo je větší než specifikovaná konstanta *CR,* pak souřadnice rozhodovací proměnné potomka x_j je rovna souřadnici rozhodovací proměnné vybraného rodiče $_{(i)}x_j$.

Druhou možností je nevytvářet kopii rodiče, ale postupně plnit (pomocí funkce **AddListItem**) na začátku prázdného jedince (vektor) souřadnicemi rozhodovacích proměnných rodiče, pokud je splněna podmínka

pravděpodobnosti náhrady souřadnice. Pokud podmínka splněna není, pak se bude jedinec plnit souřadnicemi zmutovaného jedince.

Pokud žádná souřadnice potomka x_j nebyla během tohoto procesu přepsána souřadnicí zmutovaného jedince ${}_{(\mathrm{New})}x_j$ nebo při volbě *CR* = 0, pak se nahrazuje jedna náhodně vybraná *r*-tá souřadnice rozhodovací proměnné potomka x_j (kde $r \in j$), souřadnicí rozhodovací proměnné zmutovaného jedince ${}_{(\mathrm{New})}x_j$. [14]

$$x_j = \begin{cases} {}_{(\mathrm{New})}x_j & \text{jestli } \left(R_j \leq CR\right) \vee (j = r) \\ {}_{(i)}x_j & \text{jinak} \end{cases} \tag{4.34}$$

kde:

$$\mathbf{X} = \left[x_j\right] \forall x_j \in X_j \forall j : j = \{0,1,2,\dots,n-1\} \tag{4.35}$$

$$\mathbf{X}_i = \left[{}_{(i)}x_j\right] \forall {}_{(i)}x_j \in X_j : i = \{0,1,2,\dots,m-1\}, j = \{0,1,2,\dots,n-1\} \tag{4.36}$$

$$\mathbf{X}_{\mathrm{New}} = \left[{}_{(\mathrm{New})}x_j\right] \forall {}_{(\mathrm{New})}x_j \in X_j : j = \{0,1,2,\dots,n-1\} \tag{4.37}$$

$$R_j \in (0,1) \subseteq \mathbb{R} \tag{4.38}$$

$$CR \in [0,1] \subseteq \mathbb{R} \tag{4.39}$$

$$j \in [0, n-1] \subseteq \mathbb{N} \tag{4.40}$$

$$r \in j \in [0, n-1] \subseteq \mathbb{N} \tag{4.41}$$

kde:

- $\mathbf{X}$ … Potomek vzniklý křížením nově vytvořeného jedince a rodiče.
- x_j … Souřadnice rozhodovací proměnné potomka.
- $\mathbf{X}_{\mathrm{New}}$ … Nově vytvořený jedinec pomocí mutace, který bude použit pro křížení.
- ${}_{(\mathrm{New})}x_j$ … Souřadnice rozhodovací proměnné nově vytvořeného jedince mutací.
- $\mathbf{X}_i$ … Rodič vybraný z populace rodičů s *i*-tým indexem tj. $X_{\mathrm{Pop}}[i]$.
- ${}_{(i)}x_j$ … Souřadnice rozhodovací proměnné rodiče.
- R_j … Náhodné reálné číslo pro *j*-tou rozhodovací proměnnou.
- CR … Pravděpodobnost nahrazení souřadnic rozhodovacích proměnných potomka.
- j … Index rozhodovací proměnné.
- r … Náhodně zvolený index rozhodovací proměnné potomka, který bude přepsán souřadnicí rozhodovací proměnné nově vygenerovaným jedincem pro křížení.
- n … Rozměr prohledávaného prostoru – počet rozhodovacích proměnných.
- $\mathbb{R}$ … Obor reálných čísel.
- $\mathbb{N}$ … Obor přirozených čísel.

V algoritmu může být uplatněna adaptivita pomocí mutačního parametru ω podle Ali a Törna, které bude přepočteno v každém iteračním kroku – vygenerování potomka. Pravidlo lze popsat takto: [14]

$$\omega = \begin{cases} \max\left\{\omega_{\min}, 1 - \left|\frac{F_{\max}}{F_{\min}}\right|\right\} & \text{jestli } \left|\frac{F_{\max}}{F_{\min}}\right| < 1 \\ \omega_{\min}, 1 - \left|\frac{F_{\min}}{F_{\max}}\right| & \text{jinak} \end{cases} \tag{4.42}$$

kde:

- ω … Parametr adaptivního pravidla.
- $\omega_{\min}$ … Konstanta adaptivního pravidla, která zabezpečuje aby $\omega \in [\omega_{\min}, 1)$, $\omega_{\min} > 0$, doporučená hodnota $\omega_{\min} \geq 0.45$.
- $F_{\max}$ … Největší hodnota účelové funkce jedince, ze všech jedinců v populaci.
- $F_{\min}$ … Nejmenší hodnota účelové funkce jedince, ze všech jedinců v populaci.

Předpokládá se, že adaptivní pravidlo udržuje v počátečním stádiu diverzitu prohledávání - **explorace** a v pozdějším stádiu zvyšuje intenzitu prohledávání – **exploatace**. Díky tomu roste spolehlivost hledání a rychlost konvergence.

Výhodou algoritmu je jeho jednoduchost implementace a výpočetně nenáročné generování – křížení – nového potomka. Nevýhodou diferenciální evoluce je poměrně vysoká citlivost na nastavení vstupních parametrů algoritmu ω a *CR*.

Storn a Price doporučují volit hodnoty:

- m …Velikost populace rodičů X_{Pop} tj. délka seznamu $m = \mathrm{Length}(X_{\mathrm{Pop}})$, doporučená hodnota $m = 10 \cdot n$
- ω … Parametr adaptivního pravidla, doporučená hodnota $\omega = 0.8$.
- CR … Pravděpodobnost nahrazení souřadnic rozhodovacích proměnných, doporučená hodnota $CR = 0.5$.

Je doporučeno, že tyto hodnoty se mají na základě empirických zkušeností během hledání modifikovat. Záleží zde především na zkušenosti a dobré intuici uživatele. Sami autoři ve svých testovacích úlohách užívají pro různé úlohy velmi odlišné hodnoty vstupních parametrů:

- $0.5 \leq \omega \leq 1$.
- $0 \leq CR \leq 1$.
- $m < 10 \cdot n$.

5 Teoretická východiska z hodnocení současného stavu

U řady dodávaných simulačních optimalizátorů, které mají za cíl určit vhodné nastavení vstupních parametrů simulačního modelu (rozhodovacích proměnných) tak, aby byla optimalizována činnost simulovaného systému, je ve většině případů problémem, že se optimalizátory chovají jako „černá skříňka - Black-Box“. Tyto simulační optimalizátory bývají součástí simulačních softwarů a nelze s přesností určit, které optimalizační metody byly při optimalizačním procesu použity, jaké efektivity při hledání optima dosahovaly, jaké bylo provedeno nastavení parametrů optimalizační metody atd. Proto byly na základě výše uvedené analýzy současného stavu problematiky (viz též rešerše v práci ke státní doktorské zkoušce [20]), vytipovány základní problémy běžně užívaných simulačních optimalizátorů, jejichž odstranění by mohlo vést k efektivnějšímu provádění optimalizačních experimentů na simulačních modelech za účelem nalezení optima. Těmito problémy jsou:

- Nelze externě zvolit použití vybrané optimalizační metody.
- Nelze nastavit parametry optimalizační metody (parametry optimalizačních metody jsou spravovány vnitřním algoritmem optimalizátoru).
- Dochází k přepínání metod během optimalizace a tak není možné identifikovat, která metoda byla použita a jak úspěšně (dochází k přepínání metod např. pomocí algoritmu se soutěžícími heuristikami, umělé inteligence – neuronové sítě, atd.).
- Obvykle nelze implementovat vlastní metody nebo modifikovat implementované metody, zejména z právních důvodů (většina simulačních optimalizátorů využívá stejných optimalizačních metod).
- Nelze specifikovat vlastní kritéria ukončení optimalizačního běhu.
- Potřeba vytvoření modelu v dodávaném simulačním software, nelze použít simulační model vytvořený v jiném simulačním software.
- Simulační optimalizátory nejsou vhodné pro testování optimalizačních metod na simulačních modelech (uzavřený systém).
- Nemožnost vytváření znalostní databáze simulačního modelu (použití stejného simulačního modelu vícekrát – urychlení testování pomocí nalezení výsledku simulačního experimentu v databázi; použití při vlastním vyhodnocování optimalizačních metod na simulačních modelech).
- Lze obtížně realizovat zákaznicky požadovanou vizualizaci experimentů ve 3D grafu účelové funkce – průběh hledání globálního extrému účelové funkce.
- Obtížné, spíše neproveditelné, použití na jiných IT prostředcích, včetně paralelizace simulačních běhů.

Na základě dosavadního stavu poznání a předchozího výčtu problémů standardních simulačních optimalizátorů využívaných pro diskrétní simulaci výrobních systémů a výrobních procesů ve strojírenství, byly specifikovány následující teze pro tuto práci:

- Lze dosti objektivně hodnotit efektivitu optimalizačních metod pro různé typy účelových funkcí používaných pro simulační úlohy.
- Je možné experimentálně zkoumat optimalizační algoritmy a vhodné nastavení jejich parametrů v závislosti na typu účelové funkce.
- Lze vytvořit a ověřit metodiku pro hledání co nejlepších optimalizačních algoritmů efektivně použitelných v simulaci konkrétní diskrétní výroby ve strojírenství.
- Pro testování chování optimalizačních metod bude zapotřebí analyzovat velké množství dat (odhadem v této práci cca 4 miliardy simulačních experimentů, miliony optimalizačních experimentů). Proto je žádoucí vytvořit simulační optimalizátor jako otevřenou softwarovou aplikaci s možností testovat a efektivně vyhodnocovat chování optimalizačních metod včetně možnosti je doplňovat nebo modifikovat a nastavovat jejich parametry (tj. vytvořit experimentální základnu pro experimentální zkoumání optimalizačních metod, která bude vybudována na výše uvedené metodice).
- Některé optimalizační metody jsou dosti obecně použitelné pro různé modely a typy účelových funkcí užívaných při simulaci diskrétních výrob.

6 Cíle disertační práce

Na základě hodnocení problematiky v předchozí kapitole byly stanoveny následující cíle disertační práce, jejíž výsledky by mohly vést k efektivnějšímu provádění optimalizačních experimentů na simulačních modelech (zejména pro diskrétní výrobní systémy):

- Analyzovat, vybrat a modifikovat heuristické optimalizační metody pro diskrétní simulaci.
- Vypracovat **metodiku** vyhodnocování optimalizačních metod (algoritmů) s použitím vytvořené softwarové aplikace.
- Navrhnout a implementovat otevřenou softwarovou aplikaci pro testování a používání optimalizačních metod, která bude tvořena:
    - Experimentální základnou pro ověřování vybraných optimalizačních metod.
    - Simulačním optimalizátorem.
- Testovat vybrané optimalizační metody v experimentální základně (provádět hromadné série optimalizačních experimentů), a to na relevantních simulačních modelech a účelových funkcích.
- Vyhodnotit podle metodiky výsledky rozsáhlých optimalizačních experimentů, zhodnotit testované optimalizační metody a doporučit nastavení jejich parametrů.

Definované cíle zejména reflektují nedostatky běžně užívaných simulačních optimalizátorů, které díky právním omezením nemohou dostatečně reflektovat výzkumné potřeby jak v akademické sféře, tak i sféře praktické. V průběhu několika předchozích let byla na pracovišti KPV řešena řada simulačních úloh pro průmyslovou praxi. Bohužel u většiny případů těchto úloh nebylo možné použít integrovaný simulační optimalizátor, který byl součástí simulačního systému, ve kterém byl zpracován diskrétní simulační model. Důvodem bylo především to, že simulační optimalizátor umožňuje nastavovat jen vybrané typy parametrů simulačního modelu. Toto omezení je však pro praktické úlohy těžko přijatelné.

Uzavřenost simulačních optimalizátorů vede také k tomu, že nelze porovnat efektivitu užitých optimalizačních metod na různých typech simulačních modelů, které se od sebe liší průběhem účelové funkce v prohledávaném prostoru. Převážná většina v literatuře prezentovaných analýz provedená za účelem testování optimalizačních metod je prováděna především u funkcí s různými stupni obtížnosti a je vyjádřena pouze charakteristikou standardní odchylky nalezeného optima od skutečného globálního optima. Není však příliš brán zřetel na to, že praxe namísto matematické přesnosti bude spokojena i s přijatelným řešením, které bude blízko globálního, namísto přesného určení globálního optima. Disertační práce je také proto zaměřena na návrh metodiky, která bude schopna vyhodnotit jednotlivé aspekty z pohledu úspěšnosti/neúspěšnosti při nalezení globálního optima, různých typů odchylek nalezených výsledků od globálního optima a náročnosti při hledání globálního optima (počet ohodnocení účelové funkce) u provedených optimalizačních experimentů na vybraných diskrétních simulačních modelech. Dále by také mělo být navrženo uživatelsky efektivní a přátelské grafické vyhodnocování výsledků prováděných optimalizačních experimentů (pro výzkumné aktivity v problematice).

Cílem disertační práce je také pokusit se vytipovat efektivní obecně použitelné optimalizační metody, bez ohledu na typ simulačního modelu, potažmo jeho průběhu účelové funkce, které budou také málo náchylné na nastavení vlastních parametrů optimalizační metody. U metod, jejichž efektivita při hledání globálního optima bude závislá na nastavení parametrů optimalizační metody, experimentálně stanovit vhodná nastavení jednotlivých parametrů pro různé typy účelových funkcí.

7 Použité vědecké metody zkoumání

Při zpracování této disertační práce byly využívány vědecké metody, které jsou obecně využitelné v odlišných případech, díky čemuž mohly být využity i v souvislosti s problematikou řešenou v této disertační práci.

Abstrakce jako myšlenkový proces, v jehož rámci se u různých objektů vydělují pouze jejich podstatné charakteristiky (nepodstatné se neuvažují), čímž se vytváří model objektu obsahující jen ty charakteristiky či znaky, jejichž zkoumání nám umožní získat odpovědi na otázky, které si klademe. [30]

Na základě provedené **abstrakce** problematiky simulační optimalizace byly stanoveny obecné prvky pro řešení simulačních optimalizačních úloh (rozhodovací proměnné, odezvy, účelová funkce, omezení, optimalizační metody, seznam atd.).

Za účelem užití globálních optimalizačních metod v rámci diskrétní simulační optimalizace, musely být také abstrahovány jednotlivé základní prvky týkající se globální optimalizace, např. minimalizace/maximalizace účelové funkce, iterace, kritérium ukončení atd. Dále byly abstrahovány základní problémy týkající se možných problémů globální optimalizace, které mohou nastat při experimentování, např. předčasná konvergence, multimodálnost účelové funkce atd. Popsané problémy se vyskytují v převážné většině optimalizačních úloh. Všechny vyjmenované předchozí podstatné charakteristiky obou oblastí byly popsány v rešeršní části disertační práce.

Analýza je proces faktického nebo myšlenkového rozčlenění celku (jevu, objektu) na části. Je to rozbor vlastností, vztahů, faktů postupující od celku k částem. Analýza umožňuje odhalovat různé stránky a vlastnosti jevů a procesů, jejich stavbu, vyčleňovat etapy, rozporné tendence apod. Analýza umožňuje oddělit podstatné od nepodstatného, odlišit trvatlé vztahy od nadhodilých. [30]

Samotná analýza byla použita již při samotném zpracování současného stavu řešené problematiky. Jednalo se o analýzu základních problémů běžně používaných simulačních optimalizátorů, které vyústily v kritickou rešerši „bílých míst“ v současném stavu poznání dané problematiky, jejichž odstranění by mohlo vést k efektivnějšímu provádění optimalizačních experimentů na simulačních modelech za účelem nalezení optima. Tento celkový cíl by měl být realizován pomocí vytvoření vlastní otevřené aplikace, na níž bude možné provádět simulační experimenty na vybraných diskrétních simulačních modelech.

Modelovány byly typické situace z oblasti strojírenské výroby. Muselo být provedeno jisté zjednodušení simulačního modelu, které redukuje reálný modelovaný systém (počet pozorovaných proměnných, počet analyzovaných vztahů mezi nimi, časová množina událostí, škálovatelnosti (metrika) proměnných a vztahů mezi nimi).

Aplikace využívá optimalizační metody globální optimalizace. Bylo tedy zapotřebí nejprve **identifikovat** vhodné optimalizační metody, **analyzovat** a dále je **modifikovat** takový způsobem, aby byly využitelné v rámci diskrétní simulační optimalizace. Modifikované optimalizační metody byly **implementovány** do prostředí aplikace simulační optimalizace, dále byly **testovány** a **validovány** pomocí vybraných simulačních modelů. Díky dosaženým výsledkům simulační optimalizace lze **zpětnovazebním způsobem** specifikovat případné modifikace a správné nastavení parametrů optimalizačních metod tak, aby se výsledné navržené algoritmy chovaly dle specifikovaných požadavků.

Aplikace systémového přístupu

Systémový přístup znamená, že na předmět našeho zájmu nahlížíme jako na množinu prvků a vazeb a zvažujeme všechny jeho děje a části ve významných souvislostech. Vlastnosti prvků a vazeb mezi nimi určují vlastnosti (chování) celku. [31]

Návrh samotné aplikace vycházel z analýzy potřeb simulační optimalizace prováděné na vytvořených diskrétních simulačních modelech. Jednotlivé abstrahované prvky byly začleněny do tohoto systému takovým způsobem, aby neztratily svoji obecnost.

Systematicky je pojata i tato předkládaná disertační práce. Pomocí rešerše byly definovány základní potřebné vstupy ke splnění cíle. Byla zde také popsána jednotlivá vnitřní struktura systému a jeho jednotlivých částí, včetně vazeb mezi nimi. Výstupem jsou poznatky a doporučení z analýzy provedených simulačních experimentů na diskrétních simulačních modelech.

Syntéza znamená postupovat od částí k celku. Dovoluje poznávat objekt jako jediný celek. Je to spojování poznatků získaných analytickým přístupem. [30]

Při vytváření jednotlivých částí programu je třeba neustále myslet na funkci vytvářeného programu, ale také jak tato funkčnost bude ovlivňovat celek. Je zejména důležité modifikovat jednotlivé části kódu takovým způsobem, aby vedly k efektivnějšímu chování celého programu. Rozdělením systému na dílčí subsystémy (rozdělení na jednotlivé třídy, knihovny atd.), lze efektivněji modifikovat chování celého programu. Tyto vytvořené části byly složeny do výsledného celku, v našem případě do podoby aplikace simulační optimalizace.

Syntéza byla zejména využita v oblasti optimalizačních algoritmů. Spojením poznatků chování jednotlivých částí optimalizačního algoritmu (např. selekce, křížení, mutace apod.) při různém nastavení jednotlivých parametrů těchto částí, lze predikovat výsledné chování optimalizačního algoritmu.

Analogie je založena na přenosu závěrů o platnosti určitého znaku jednoho objektu na objekt jiný, vycházející ze zjištění příbuznosti obou objektů podle jiného znaku. [30]

Analogie byla využita zejména při modifikaci jednotlivých optimalizačních metod. Optimalizační metody, implementované ve formě optimalizačních algoritmů, byly transformovány s využitím analogie fyzikálních a biologických principů uplatňovaných v přírodě (např. namísto generování jediného prvku v některých pseudogradientních metodách bylo využito generování celé populace, což mělo omezit vliv předčasné konvergence do lokálního extrému, mutace složek - rozhodovacích proměnných - prvku).

Tvůrčí metody

Cílem metod tvůrčího myšlení je zvýšit pravděpodobnost úspěšného vyřešení problému v průběhu tvůrčího procesu. [30]

Při vytváření metodiky pro hodnocení nebylo postupováno standartním způsobem, tradičně užívaném v rámci optimalizačních úloh globální optimalizace, ale byla specifikována nová metodika, která zahrnuje odlišná kritéria, která mohou být využitelná při porovnání efektivnosti optimalizačního algoritmu při hledání globálního optima. Jedná se o začlenění ohodnocení statistických charakteristik, přístupů jak definovat vhodná nastavení parametrů optimalizačního algoritmu apod.

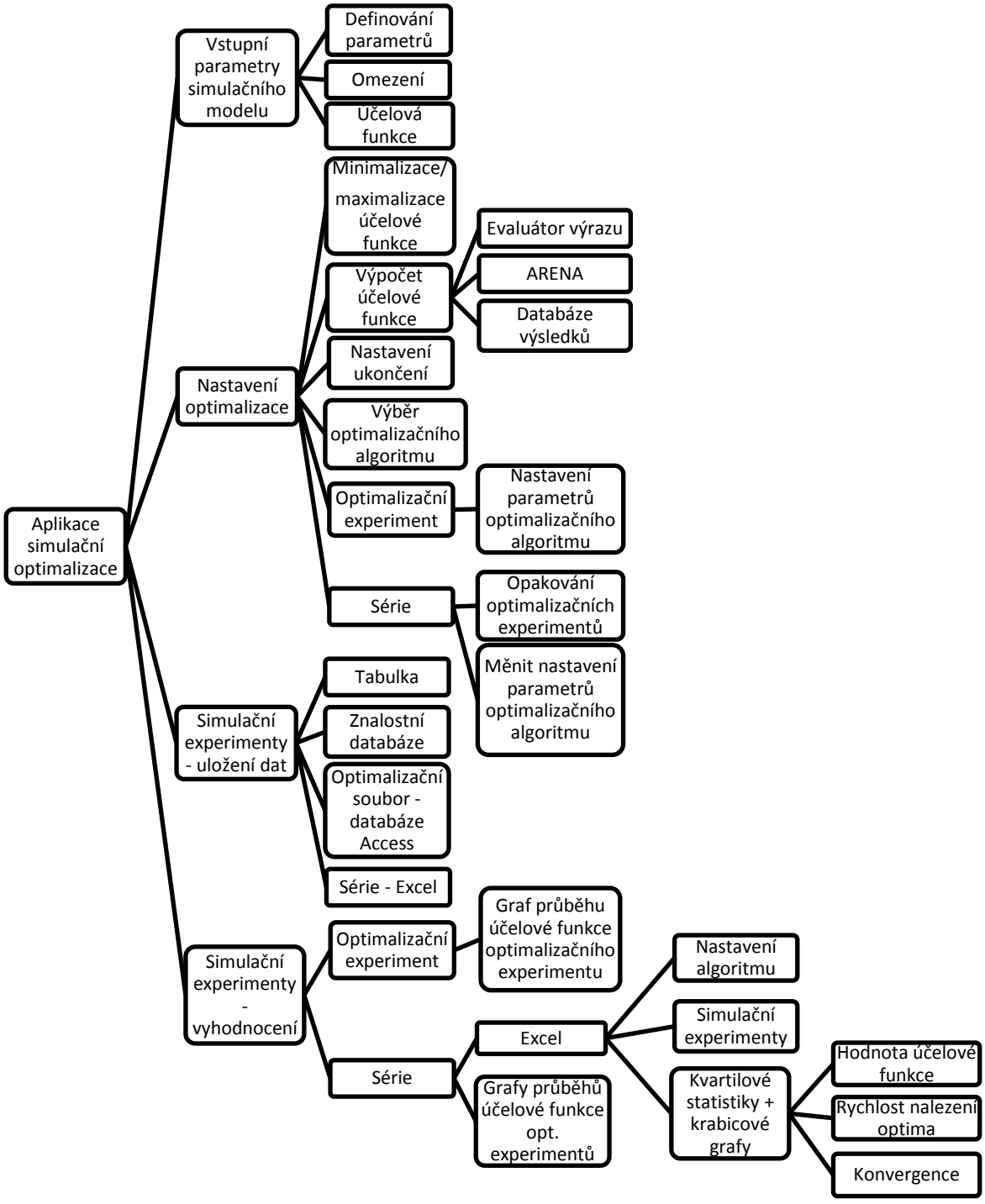
Metoda porovnávání

Porovnávání umožňuje určit to, co je společné dvěma jevům nebo naopak, co je odlišuje. Objekty je nutné srovnávat podle nejdůležitějších znaků, podstatných z hlediska zkoumané stránky objektu. [30]

Cílem porovnání bylo určení podstatných rysů různých typů optimalizačních algoritmů v závislosti na průběhu účelové funkce. Bylo snahou vyvodit obecný závěr na základě poznatků provedené analýzy chování optimalizačního algoritmu na různých typech průběhů účelové funkce – **indukce**.

8 Vývoj softwarové aplikace simulační optimalizace

Za účelem testování chování vybraných modifikovaných optimalizačních metod na vytvořených diskrétních simulačních modelech, hromadného provádění sérií (opakování optimalizačních experimentů s konkrétním nastavením parametrů optimalizačních algoritmů s cílem určit vhodné nastavení jejich parametrů v závislosti na průběhu účelové funkce) a validace navržené metodiky vyhodnocení optimalizačních experimentů, bylo zapotřebí vyvinout, tj. navrhnout, vytvořit a ověřit vlastní aplikaci simulační optimalizace, která by disponovala potřebnými funkcemi pro plnění vytýčeného cíle. Po provedení analýzy bylo navrženo následující schéma zahrnující potřebné funkce, které jsou implementovány v aplikaci simulační optimalizace – viz Obr. 8-1.



Obr. 8-1 Funkce aplikace simulační optimalizace

8.1 Implementované optimalizační metody

Ve vytvořené aplikaci simulační optimalizace bylo implementováno celkem 8 optimalizačních metod. Tyto metody byly modifikovány za účelem využití v rámci diskrétní simulace (transformace nepřípustných řešení pomocí metod práce s omezujícími podmínkami, vygenerování přípustného řešení, atd.).

Algoritmy včetně funkcí a procedur jsou uvedeny v příloze ve formě pseudopascalských algoritmů (viz **příloha** - kapitola 4.1 Implementované optimalizační metody).

Na základě provedené rešerše bylo také provedeno sjednocení různých variant optimalizační metody do jedné souhrnné optimalizační metody. Uživatel tedy může kombinovat různé přístupy pomocí nastavení parametrů optimalizačního algoritmu.

Dále je uvedena stručná charakteristika jednotlivých algoritmů.

8.1.1 Náhodné prohledávání

Generování souřadnic rozhodovacích proměnných prvku se děje na základě rovnoměrného rozdělení v celém prohledávaném prostoru. Algoritmus byl odvozen od metody „Náhodné prohledávání“ (viz kapitola 4.2.1). Algoritmus je využitelný zejména v případech, kdy nemáme žádnou apriorní znalost průběhu účelové funkce, ale jsme schopni provést větší počet simulačních experimentů.

8.1.2 Downhill Simplex

Algoritmus pracuje s množinou bodů označovanou jako „Simplex“. Tato množina bodů obsahuje minimálně $n+1$ bodů, kde n vyjadřuje rozměr prostoru rozhodovacích proměnných. Tyto body jsou nekomplanární (lineárně nezávislé body). Základními fázemi algoritmu jsou reflexe, expanze, kontrakce a redukce. Algoritmus byl odvozen od metody „Downhill Simplex - Simplexová metoda“ (viz kapitola 4.3.1).

8.1.3 Stochastický horolezecký algoritmus

Generování prvků je prováděno v sousedství nejlepšího nalezeného prvku z předchozích vygenerovaných prvků (**populace**). Generování je prováděno pomocí transformací souřadnic (**mutací**) nejlepšího nalezeného prvku z předchozí populace**,** tj., generování náhodného čísla podle rovnoměrného rozdělení v rozmezí délky intervalu dolní a horní meze definovaného pro každou jednotlivou rozhodovací proměnnou. Algoritmus byl odvozen od metody „Stochastický horolezecký algoritmus “ (viz kapitola 4.5.1).

8.1.4 Stochastické zakázané prohledávání

Během optimalizačního procesu je každý nově vygenerovaný prvek vložen do seznamu zakázaných prvků. Takový prvek nesmí být navštíven, pokud nesplňuje aspirační kritérium. Pokud je překročena povolená délka seznamu zakázaných prvků, je provedeno vyjmutí prvku z tohoto seznamu – metoda FIFO. Generování prvku v sousedství dosud nejlepšího nalezeného prvku z předchozích prvků (populace), se děje pomocí transformace (mutace) nejlepšího prvku z předchozích prvků na základě rovnoměrného rozdělení. V aplikaci je implementována možnost volby, že nové prvky (populace) budou vytvořeny na základě nejlepšího prvku z přechozích prvků (prvky v poslední populaci), nebo budou generovány na základě dosud nejlepšího nalezeného prvku. Algoritmus byl odvozen od metody „Metoda zakázaného prohledávání“ (viz kapitola 4.5.2).

8.1.5 Stochastické simulované žíhání

Generování prvku v sousedství dosud nejlepšího nalezeného prvku se děje pomocí transformace (mutace) prvku z předchozí iterace. Tato transformace může být prováděna buď pomocí změny náhodně vybrané jedné složky - rozhodovací proměnné - předchozí prvku, nebo pomocí změny u všech složek prvku (změna všech rozhodovacích proměnných). Náhodné číslo vyjadřující složku nově generovaného prvku - rozhodovací proměnnou – může být generováno s použitím specifikovaného rozptylu nebo může být generováno v rozsahu intervalu příslušné rozhodovací proměnné. Horší prvek je přijat jen s jistou pravděpodobností, která je závislá na parametru metody - teplota. Teplota je při přijetí horšího prvku snižována (v algoritmu simulovaného žíhání je implementována možnost snižovat teplotu tehdy, pokud je náhodně vygenerované číslo menší než

pravděpodobnost přijetí horšího řešení nebo teplota bude snižována vždy, pokud bude vygenerován horší prvek). Pokud dojde ke snížení teploty pod specifikovanou minimální hranici, teplota je opětovně zvýšena na počáteční definovanou hodnotu. Algoritmus byl odvozen od metody „Stochastické simulované žíhání“ (viz kapitola 4.5.3)

8.1.6 Stochastické lokální prohledávání

Generování prvku je prováděno v sousedství dosud nejlepšího nalezeného prvku (mutace). Algoritmus byl odvozen od metody „Stochastické lokální prohledávání“ (viz kapitola 4.5.4).

8.1.7 Evoluční strategie

Algoritmus evoluční strategie využívá postupnou vývojovou strategii $(\mu + \lambda)$, tj. generace potomků je tvořena určitým počtem jedinců s nejlepšími funkčními hodnotami ze všech jedinců jak rodičovské populace, tak populace potomků. Generování jedince je provedeno pomocí mutace původního jedince na základě normálního rozdělení - implementováno Rechenbergovo adaptivní pravidlo jedné pětiny úspěšnosti upravující hodnoty směrodatných odchylek zvlášť pro každou dimenzi na základě vypočtené relativní četnosti úspěchů - intenzifikace vs. diverzifikace. Pokud vypočtená relativní četnost úspěchů byla shodná s definovanou konstantou, směrodatné odchylky se nemění. Přiřazení fitness jedincům je provedeno na základě setříděné populace podle hodnoty účelové funkce. V algoritmu je použita turnajová selekce. Algoritmus byl odvozen od metody „Evoluční strategie“ (viz kapitola 4.7).

8.1.8 Diferenciální evoluce

Selekce je provedena mezi rodičem a jeho potomkem, který vznikl křížením rodiče s nově vytvořeným jedincem vzniklým mutací jedinců pomocí metody BEST (náhodně čtyři vybraní jedinci, kteří jsou různí od nejlepšího jedince v populaci a aktuálního vybraného jedince z populace).

Lepší jedinec, z těchto dvou porovnávaných jedinců, je následně zařazen do populace, která kompletně nahradí populaci rodičů. V algoritmu je uplatněno adaptivní pravidlo podle Ali a Törna, která upravuje hodnotu mutačního parametru, v aplikaci označený jako Koef_F. Dále je definována pravděpodobnost určená pro křížení, tzn., že potomek zdědí gen zmutovaného jedince nebo zdědí gen po rodiči. V aplikaci je tento parametr označený jako Koef_C. Algoritmus byl odvozen od metody „Diferenciální evoluce“ (viz kapitola 4.8)

Následující tabulka (viz Tabulka 8-1) obsahuje názvy optimalizačních algoritmů včetně parametrů jednotlivých optimalizačních metod a popisu jejich podstaty.

Optimalizační metoda	Optimalizační algoritmus	Parametr algoritmu	Popis
Náhodné prohledávání	Random Search	Generovat stejné prvky (Ano/Ne)	Umožnění generování stejných prvků během optimalizačního procesu
Downhill Simplex - Simplexová metoda	Downhill Simplex	Reflexe	Koeficient ovlivňující vzdálenost při překlopení nejhoršího prvku simplexu přes těžiště simplexu. Tím je získán prvek reflexe
		Expanze	Koeficient ovlivňující vzdálenost při překlopení prvku získaného pomocí reflexe simplexu, a to ve směru, který prochází spojnicí nejhoršího prvku a těžiště. Tím je získán prvek expanze
		Kontrakce	Koeficient ovlivňující vzdálenost umístění prvku kontrakce mezi prvek reflexe a těžiště simplexu
		Redukce	Koeficient ovlivňující velikost smrštění všech bodů simplexu směrem k nejlepšímu bodu simplexu
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení nebo prvek má být specifikován jako počáteční přípustné řešení

Stochastický horolezecký algoritmus	Hill Climbing	Rozptyl	Vymezení délky hranice oblasti kolem vybraného prvku, ve kterém se budou generovat další prvky
		Velikost populace	Počet prvků, které budou generovány v oblasti kolem vybraného prvku
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení, nebo prvek má být specifikován jako počáteční přípustné řešení
Stochastické zakázané prohledávání	Tabu Search	Rozptyl	Vymezení délky hranice oblasti kolem vybraného prvku, ve kterém se budou generovat další prvky
		Velikost populace	Počet prvků, které budou generovány v oblasti kolem vybraného prvku
		Tabu Length	Počet předcházejících prvků, které jsou umístěny v seznamu zakázaných prvky, tj. tyto prvky nesmí být znovu navštíveny (s výjimkou aspiračního kritéria)
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení nebo prvek má být specifikován jako počáteční přípustné řešení
		Optimalizace podle poslední populace (Ano/Ne)	Nové prvky (populace) budou vytvořeny na základě nejlepšího prvku z přechozích prvků (prvky v poslední populaci) nebo budou generovány na základě dosud nejlepšího nalezeného prvku
Simulované žíhání	Simulated Annealing	Rozptyl	Vymezení délky hranice oblasti kolem vybraného prvku, ve kterém se budou generovat další prvky
		Měnit jen jeden parametr (Ano/Ne)	Generování nového prvku bude provedeno pomocí změny náhodně vybrané jedné složky - rozhodovací proměnné - předchozí prvku nebo generování nového prvku bude provedeno pomocí změny u všech složek prvku (změna všech rozhodovacích proměnných)
		Beta	Nezáporná konstanta určující výši poklesu teploty při přijetí horšího řešení (snížení pravděpodobnosti přijetí horšího řešení), $\beta \in (0,1)$
		Minimální teplota	Výše teploty, při které dojde k opětovnému nastavení na počáteční teplotu
		Generovat dle rozptylu (Ano/Ne)	Náhodné číslo vyjadřující složku nově generovaného prvku - rozhodovací proměnnou - bude generováno pomocí specifikovaného rozptylu nebo bude generováno v rozsahu intervalu příslušné rozhodovací proměnné
		Snižovat teplotu jen při přijetí horšího řešení (Ano/Ne)	Snížení teploty tehdy, pokud je náhodně vygenerované číslo menší než specifikovaná pravděpodobnost přijetí horšího řešení nebo teplota bude snížena vždy, pokud dojde k vygenerování horšího prvku než je předchozí prvek
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení nebo prvek má být specifikován jako počáteční přípustné řešení
Stochastické lokální prohledávání	Local Search	Rozptyl	Vymezení délky hranice oblasti kolem vybraného prvku, ve kterém se budou generovat další prvky
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení nebo prvek má být specifikován jako počáteční přípustné řešení

Evoluční strategie	Evolution Strategy	Velikost populace	Určuje počet nejlepších jedinců, kteří budou umístěni do výsledné populace. Výběr je prováděn z populace, která vznikla spojením populace rodičů a populace potomků
		Mmp	Určuje počet, kolik bude vygenerováno potomků
		q	Parametr určuje, kolik má být sledováno předcházejících úspěchů/neúspěchů u jedinců → informace pro adaptivní pravidlo, které určuje velikost okolí
		k	Počet jedinců v turnajové selekci
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení, nebo prvek má být specifikován jako počáteční přípustné řešení
Diferenciální evoluce	Differential Evolution	Velikost populace	Určuje počet jedinců, kteří budou umístěni do výsledné populace přeživších jedinců na základě selekce rodič vs. potomek
		Koef_F	Koeficient využívaný při mutaci původního jedince
		Koef_C	Výše pravděpodobnosti, že potomek zdědí gen zmutovaného jedince nebo zdědí gen po rodiči
		První populace náhodně (Ano/Ne)	Generování náhodného počátečního přípustného řešení nebo jedinec má být specifikován jako počáteční přípustné řešení

Tabulka 8-1 Seznam implementovaných optimalizačních algoritmů a jejich parametrů

V aplikaci simulační optimalizace byl také implementován algoritmus **Total Search**. Algoritmus umožňuje provést kompletní prořez prohledávaného prostoru a vyhodnocení všech jeho prvků (ohodnocení prvků hodnotou účelové funkce). Toto je vhodné (pokud je reálně proveditelné) použít pro získání referenční („přesné“) hodnoty optima při testování výše uvedených optimalizačních algoritmů. V tomto algoritmu byla využita rekurze.

8.2 Popis prostředí softwarové aplikace simulační optimalizace

Aplikace simulační optimalizace byla vytvořena v prostředí jazyku Visual Basic. Tento programovací jazyk byl vybrán proto, že umožňuje napojení na simulační software ARENA a na databázi, která v sobě zahrnuje informace o rozhodovacích proměnných, účelové funkci, specifikovaných omezeních, nastavení jednotlivých optimalizačních algoritmů, kritériích ukončení atd.

Po spuštění aplikace simulační optimalizace se zobrazí hlavní panel, v němž si uživatel po rozbalení položky „Soubor“ zvolí cestu k uloženému optimalizačnímu souboru *.opt ve formě databáze (databáze aplikace Access byla zvolena z důvodu možnosti načtení vstupních dat do simulačního modelu vytvořeném v prostředí ARENA). Pomocí položky „Soubor“ si může uživatel vytvořit vlastní optimalizační soubor *.opt. Po otevření optimalizačních souboru se objeví hlavní menu, které obsahuje následující položky:

- Vstupní parametry.
- Nastavení optimalizace.
- Simulační experimenty.

8.2.1 Vstupní parametry

- Definování parametrů – slouží pro specifikaci rozhodovacích proměnných (vstupních parametrů) simulačního modelu, které budou uloženy v databázi. Každá položka pak obsahuje následující atributy (viz kapitola 3.1) - ID (j), název (X_j), dolní mez (a_j), horní mez (b_j), typ (diskrétní - celočíselné, spojité – jednotlivé hodnoty rozhodovací proměnné jsou určeny krokem v rozsahu dolní a horní meze), krok (s_j), počáteční hodnota ($^{(0)}x_j^*$ - souřadnice rozhodovací proměnné počátečního přípustného řešení),

komentář. Snímek prostředí softwarové aplikace simulační optimalizace pro specifikaci rozhodovacích proměnných je uveden v příloze (viz **příloha** – kapitola 4.2.1 Vstupní parametry, Obr. 4-1 Snímek prostředí pro specifikaci rozhodovacích proměnných).

- Omezení – sestavení vlastní funkce omezení pomocí matematických operátorů implementovaných ve formě tlačítek a seznamu vstupních parametrů simulačního modelu ($g(\mathbf{X})$ viz kapitola 3.3.1), tj., rozhodovacích proměnných uložených v databázi. Lze také vyhodnotit sestavený výraz. Snímek prostředí softwarové aplikace simulační optimalizace pro specifikaci omezení je uveden v příloze (viz **příloha** - kapitola 4.2.1 Vstupní parametry, Obr. 4-2 Snímek prostředí pro specifikaci omezení).
- Účelová funkce - specifikace vlastní účelové funkce ($F(\mathbf{X})$ viz kapitola 3.2), jejíž předpis je sestaven pomocí výběru rozhodovací proměnné ze seznamu rozhodovacích proměnných (vstupních parametrů) simulačního modelu uložených v databázi a matematických operátorů implementovaných ve formě tlačítek. Lze také vyhodnotit sestavený výraz účelové funkce (výpočet se provede bez použití simulačního nástroje ... $y = F(\mathbf{X})$). Snímek prostředí softwarové aplikace simulační optimalizace pro specifikaci účelové funkce je uveden v příloze (viz **příloha** - kapitola 4.2.1 Vstupní parametry, Obr. 4-3 Snímek prostředí pro specifikaci účelové funkce).

8.2.2 Nastavení optimalizace

- Data – horní panel nabídky:
  - Načíst dočasnou tabulku experimentování – načtení znalostní databáze provedených simulačních experimentů do paměti. Simulační experiment bude proveden tehdy, pokud nebyl nalezen v paměti, která obsahuje simulační experimenty z načtené znalostní databáze a provedené simulační experimenty uložené v optimalizačním souboru. Načítat lze:
    - S kontrolou existence stejného záznamu – při načítání je prováděna kontrola na duplicitní údaje.
    - Bez kontroly existence stejného záznamu – při načítání není prováděna kontrola na duplicitní údaje. Proces načítání je výrazně rychlejší.
  - Uložit dočasnou tabulku experimentování – uložení znalostní databáze provedených simulačních experimentů.
- Minimalizace/maximalizace účelové funkce.
- Nastavení ukončení – výběr kritéria ukončení optimalizačního experimentu (viz kapitola 3.7.3). Snímek prostředí softwarové aplikace simulační optimalizace pro specifikaci minimalizace/maximalizace účelové funkce a nastavení ukončení je uveden v příloze (viz **příloha** - kapitola 4.2.2 Nastavení optimalizace, Obr. 4-4 Snímek prostředí pro specifikaci minimalizace/maximalizace účelové funkce a nastavení ukončení). Typy ukončení mohou být:
  - Ukončení v definovaný čas.
  - Ukončení po definovaném počtu iterací.
  - Ukončení při dosažené hodnotě účelové funkce (Value To Reach).
  - Poměr zlepšení sub-optima –

$$\omega = \begin{cases} 1 - \left(\dfrac{F\left({}^{(k)}\mathbf{X}^*\right)}{F\left({}^{(k-1)}\mathbf{X}^*\right)} \right) & \text{jestli } F(\mathbf{X}) \text{ bude minimalizována} \\ \left| 1 - \left(\dfrac{F\left({}^{(k)}\mathbf{X}^*\right)}{F\left({}^{(k-1)}\mathbf{X}^*\right)} \right) \right| & \text{jestli } F(\mathbf{X}) \text{ bude maximalizována} \end{cases} \tag{8.1}$$

$${}^{(k)}\mathbf{X}^*, k = \{0,1,2, \dots, K-1\} \tag{8.2}$$

kde:

- ω … Poměr hodnoty kriteriální funkce aktuálního kandidáta řešení (nejlepšího nalezeného prvku v aktuální k-té iteraci) vůči kandidátu řešení z předchozího kola algoritmu.
- k ... Index iterace.

- K ... Celkový počet iterací.
- $F\left({}^{(k)}\mathbf{X}^{*}\right)$ … Hodnota účelové funkce nejlepšího prvku v aktuální k-té iteraci algoritmu.
- $F\left({}^{(k-1)}\mathbf{X}^{*}\right)$ … Hodnota účelové funkce nejlepšího prvku v předchozí iteraci algoritmu, tj. $k-1$-té iteraci algoritmu.

  - Rozdíl nejlepší a nejhorší hodnoty účelové funkce v iteraci – rozdíl hodnoty účelové funkce nejlepšího a nejhoršího prvku ze seznamu prvků vygerenerovaných v rámci jedné iterace (v kontextu evolučních algoritmů nejlepšího a nejhoršího jedince v populaci).
  - Maximální počet provedených pokusů na jinou hodnotu – počet provedených pokusů, kolikrát bude tolerováno vygenerování stejného prvku. Vhodné volit zejména v případech, kde hrozí, že prohledávání se zacyklí.

- Nastavení optimalizačních metod – tato záložka obsahuje seznam optimalizačních metod implementovaných ve formě algoritmů popsaných v **příloze** disertační práce (viz **příloha** – kapitola 4.1 Implementované optimalizační metody). Uživatel si vybere jeden optimalizační algoritmus, který bude použit při optimalizaci. Optimalizační algoritmy jsou: Random Search, Downhill Simplex, Local Search, Tabu Search, Simulated Annealing, Differential Evolution, Evolution Strategy. Algoritmus Total Search umožňuje kompletní prořez celého prohledávaného prostoru – všech možných kombinací hodnot rozhodovacích proměnných. Při výběru optimalizačního algoritmu lze dále nastavit parametry tohoto algoritmu. Jednotlivé parametry jsou přednastaveny na základě informací z optimalizačního souboru. Snímek prostředí softwarové aplikace simulační optimalizace pro výběr optimalizačního algoritmu, který bude automaticky nastavovat hodnoty rozhodovacích proměnných tak, aby byly minimalizovány/maximalizovány hodnoty účelové funkce je uveden v příloze. Snímek také obsahuje záložku pro nastavení parametrů vybraného optimalizačního algoritmu (viz **příloha** - kapitola 4.2.2 Nastavení optimalizace, Obr. 4-5 Snímek prostředí pro výběr optimalizačního algoritmu a nastavení jeho parametrů).
- Spuštění optimalizace - snímek záložky „Spuštění optimalizace“ softwarové aplikace simulační optimalizace pro určení výpočtu účelové funkce, tlačítek pro spuštění a zastavení optimalizace, exportu grafu účelové funkce do obrázku, informací o průběhu optimalizačního experimentu (grafu účelové funkce, tabulky vygenerovaných prvků, nejlepšího dosaženého výsledku) a dalších možností nastavení je uveden v příloze (viz **příloha** - kapitola 4.2.2 Nastavení optimalizace, Obr. 4-6 Snímek prostředí záložky „Spuštění optimalizace“).
  - Výpočet účelové funkce pomocí – v combo boxu si uživatel může vybrat ze dvou možností:
    - Vypočet pomocí účelové funkce ($y = F(\mathbf{X})$) - zda chce uživatel experimentovat na vlastní sestavené účelové funkci – předpis účelové funkce byl v tomto případě definován v části „Vstupní parametry“.
    - Simulační systém ARENA - zda chce uživatel provádět simulační experimenty s vlastním vytvořeným simulačním modelem v prostředí ARENA – v tomto případě je načtena cesta k simulačnímu modelu vytvořeném v prostředí ARENA. Hodnota účelové funkce je vypočtena na základě odezev ze simulačního modelu … $y = F(O(\mathbf{X}))$ (viz kapitola 3.2, předpis (3.7)) Uživatel má možnost cestu přenastavit kliknutím na tlačítko „Vložit“. Protože složité modely mohou zahltit paměť, kterou využívá simulační software ARENA, byla v prostředí implementována možnost „Zavírat simulační model po každém simulačním experimentu“ ve formě zaškrtávacího tlačítka. Při zaškrtnuté volbě se po dokončení simulačního běhu simulační model zavře a paměť se uvolní.
  - Spustit optimalizaci – tímto tlačítkem uživatel spouští simulační experimentování s vybraným modelem – funkce simulačního optimalizátoru. Během simulace je průběžně zobrazován stav simulačních experimentů – v levé tabulce jsou zobrazovány jednotlivé simulační experimenty s nastavenými hodnotami jednotlivých rozhodovacích proměnných (vstupní parametry simulačního modelu). V pravé části okna je průběžně vykreslován graf hodnot účelové funkce při jednotlivých simulačních experimentech. Pod těmito okny je zobrazena dosud nejlepší získaná hodnota účelové funkce a při jakém experimentu byla tato hodnota dosažena. Nalevo vedle tlačítka spustit optimalizaci jsou dvě okna, ve kterých je zobrazen předpis vlastního definovaného omezení a předpis vlastní účelové funkce. Uživatel má možnost zastavit simulační experimenty pomocí tlačítka zastavit optimalizaci. V dolní liště je uživatel informován o vybraném typu (minimalizace/maximalizace) optimalizace, zvoleném

optimalizačním algoritmu a během experimentování je průběžné zobrazována informace kolikátý simulační experiment ze všech možných experimentů (včetně prvků z oblasti nepřípustných řešení pokud je definováno omezení) probíhá.

- Spuštění sérií optimalizací – snímek záložky „Spuštění série optimalizací“ softwarové aplikace simulační optimalizace opakování optimalizačních experimentů za účelem omezení vlivu náhody na chování zvoleného optimalizačního algoritmu a statistického vyhodnocení jejího chování je uveden v příloze (viz **příloha** - kapitola 4.2.2 Nastavení optimalizace, Obr. 4-7 Snímek prostředí záložky „Spuštění série optimalizací“).
    - Stejné nastavení spustit x krát – zaškrtávací tlačítko slouží ke zvolení provádění optimalizačních experimentů v rámci jedné série. Při aktivní volbě lze nastavit počet opakování optimalizačního experimentu s konkrétním nastavením parametrů zvoleného optimalizačního algoritmu.
    - Měnit nastavení parametrů optimalizační metody – při aktivní volbě zaškrtávacího tlačítka se budou provádět optimalizační experimenty s jednotlivými konkrétními nastaveními zvoleného optimalizačního algoritmu v závislosti na nastavených hodnotách parametrů optimalizačního algoritmu v pravé tabulce. V této tabulce obsahující jednotlivé parametry zvoleného optimalizačního algoritmu je zapotřebí definovat krok u jednotlivých parametrů. Po provedení definovaného počtu opakování je hodnota příslušného parametru změněna o specifikovaný přírůstek (krok). Hodnota parametru optimalizačního algoritmu se pohybuje v rozmezí definované dolní a horní meze. Počet provedených sérií (jednotlivých nastavení parametrů optimalizačního algoritmu) bude:

$$c = \prod_{i=1}^{l} \left(\left(\frac{|b_i - a_i|}{s_i} \right) + 1 \right) \tag{8.3}$$

$$f = c \cdot r \tag{8.4}$$

kde:

- c ... Počet provedených sérií (konkrétních nastavení parametrů optimalizačního algoritmu).
- i ... Index parametru zvoleného optimalizačního algoritmu.
- a_i ... Dolní mez prohledávaného intervalu i-tého parametru zvoleného optimalizačního algoritmu.
- b_i ... Horní mez prohledávaného intervalu i-tého parametru zvoleného optimalizačního algoritmu.
- s_i ... Velikost kroku i-tého parametru zvoleného optimalizačního algoritmu.
- l ... Počet parametrů zvoleného optimalizačního algoritmu.
- f ... Počet provedených optimalizačních experimentů.
- r ... Počet opakování optimalizačních experimentů v rámci jedné série.

8.2.3 Simulační experimenty

Simulační experiment, tj. výpočet účelové funkce, probíhá za pomoci:

1. Evaluátoru, implementovaného přímo v aplikaci simulační optimalizace, který vyhodnotí uživatelem sestavený výraz účelové funkce (účelové funkce specifikována předpisem funkce).
2. Simulačního běhu na simulačním modelu vytvořeném v prostředí ARENA, díky němuž jsou získány odezvy ze simulačního modelu, které jsou následně argumentem účelové funkce, specifikované uvnitř simulačního modelu.
3. Nalezením simulačního experimentu ve znalostní databázi aplikace simulační optimalizace.

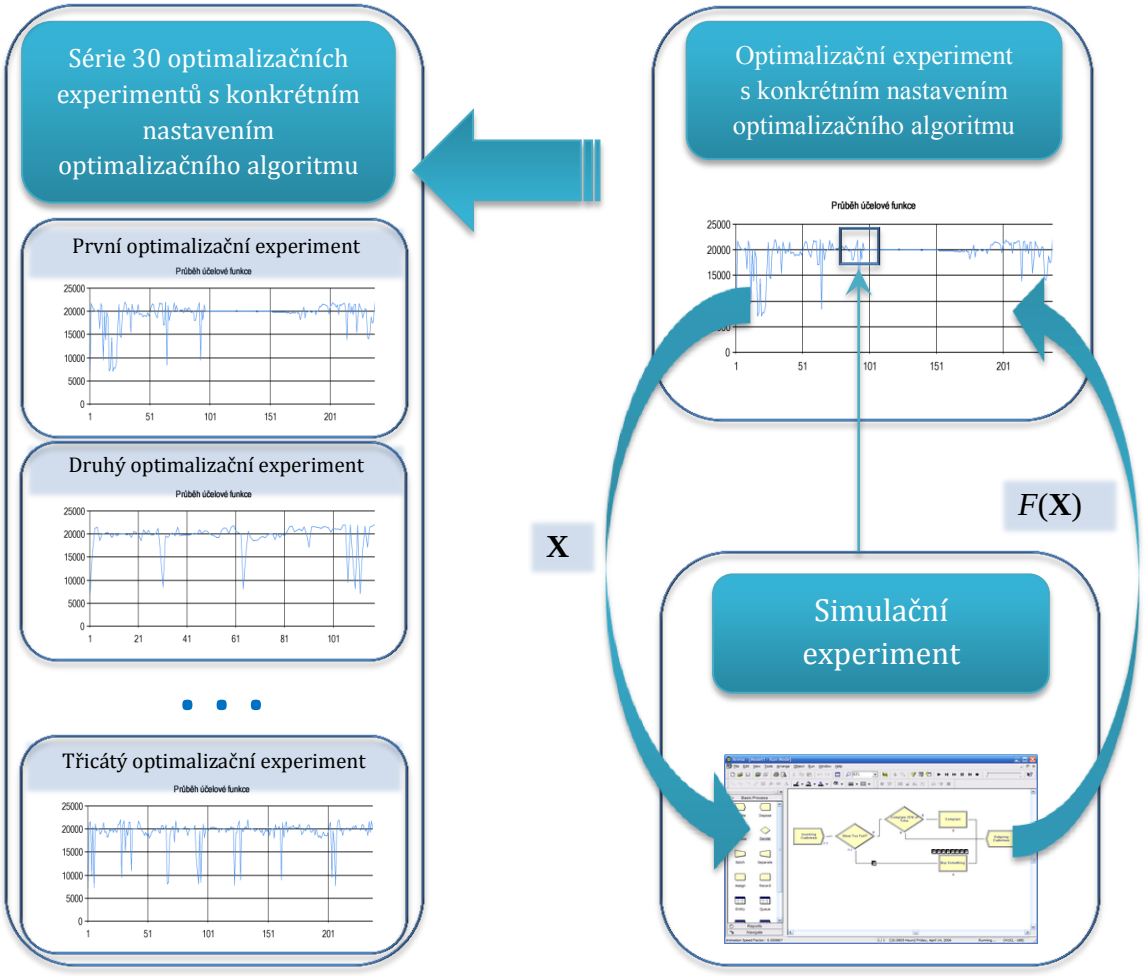
Simulační experimenty jsou v průběhu optimalizačního procesu ukládány do paměti aplikace (ve formě tabulky) a dále jsou průběžně ukládány do Accessovské databáze - optimalizačního souboru *.opt. Díky tomu uživatel může později otevřít provedený optimalizační experiment. Při otevření optimalizačního souboru je zároveň proveden import provedených simulačních experimentů do dočasné znalostní databáze spolu s vykreslením grafu průběhu účelové funkce u provedeného optimalizačního experimentu v prostředí aplikace. U tabulky výsledků jednotlivých simulačních experimentů lze také použít vzestupné nebo sestupné třídění u jednotlivých sloupců, tj., čísla provedeného experimentu, hodnot rozhodovacích proměnných a hodnot účelové funkce.

Provedené optimalizační experimenty na stejném simulačním modelu lze exportovat z dočasné paměti aplikace do znalostní databáze ve formě souboru *.xml. Takto exportovaná znalostní databáze pak může být kdykoliv importována do dočasné paměti, díky čemuž nemusí být prováděny identické simulační experimenty. Tento návrh byl realizován zejména z důvodu rychlejšího testování vlivu nastavení parametrů optimalizačního algoritmu na chování optimalizačního algoritmu při optimalizačním procesu.

V převážné části optimalizačních algoritmů je implementována náhoda. Aby bylo možné analyzovat a statisticky vyhodnocovat chování algoritmu, je vhodné stejné nastavení optimalizačního algoritmu (určité nastavení parametrů optimalizačního algoritmu) provádět několikrát – série.

Členění vzhledem k počtu prováděných simulačních experimentů je v našem případě děleno (viz Obr. 8-2) na:

1. **Simulační experiment** – simulační experiment je prováděn buď na modelu specifikované funkce, nebo modelu vytvořeném v simulačním softwaru (simulační běh). Odezvy simulačního modelu jsou argumenty účelové funkce.
2. **Optimalizační experiment** s konkrétním nastavením optimalizačního algoritmu - optimalizační proces mající za účel vyhledat optimum účelové funkce na základě automaticky řízeného opakování simulačních experimentů.
3. **Série** optimalizačních experimentů s konkrétním nastavením optimalizační metody implementované ve formě optimalizačního algoritmu – opakování optimalizačních experimentů.

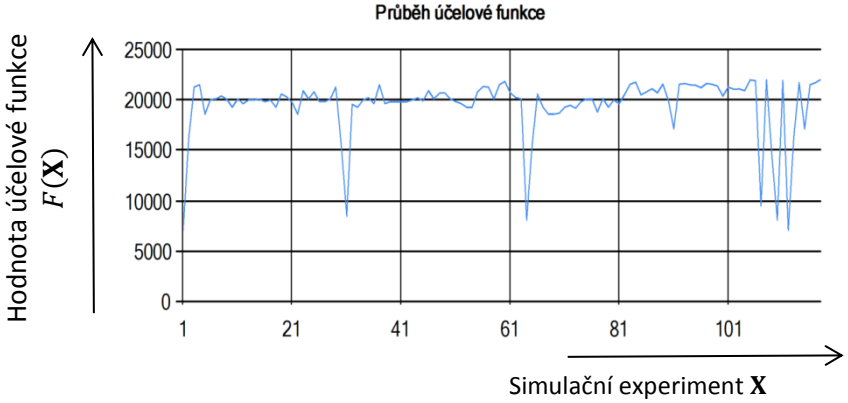


Obr. 8-2 Princip členění vzhledem k počtu prováděných simulačních experimentů

8.2.4 Vyhodnocení simulačních experimentů

Provedené simulační experimenty v rámci jednoho optimalizačního experimentu jsou průběžně vykreslovány v grafu „Průběh účelové funkce“. Tento graf zachycuje velikost hodnot účelové funkce u jednotlivých řešení -

prvků, která byla vygenerována pomocí zvoleného optimalizačního algoritmu – viz Obr. 8-3. Po ukončení optimalizačního experimentu lze graf exportovat jako obrázek typu *.png.

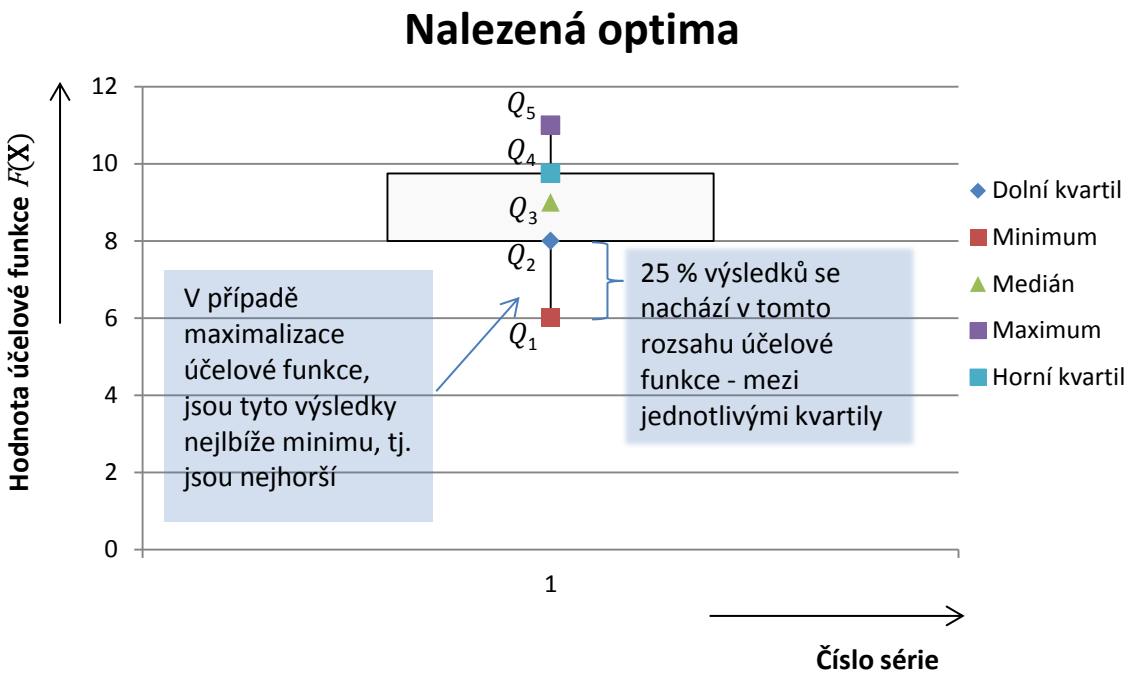


Obr. 8-3 Příklad grafu průběhu hledání optima - simulační model výrobní linky (maximalizace hodnot účelové funkce) - simulační experimenty provedené pro konkrétní nastavení optimalizačního algoritmu

Pokud jsou optimalizační experimenty prováděny v sériích, je pro každý optimalizační experiment (pomocí aplikace simulační optimalizace) exportován a uložen obrázek grafu průběhu účelové funkce. Tento detailní přístup však přestává být efektivní při větším počtu sérií nebo při větším počtu opakování optimalizačních experimentů v rámci série. Z tohoto důvodu byl využit jako vhodný způsob reprezentace vyhodnocení, krabicový graf (*Box Plot*). Krabicový graf umožňuje vykreslení rozsahu hodnot. Pomocí jednotlivých charakteristik krabicového grafu lze např. interpretovat kvalitu (posuzovaná podle hodnot účelové funkce) poskytovaných prvků (nalezených možných řešení) v sérii jako jedné „krabičky s vousy“ (krabice). Charakteristiky jsou (viz Obr. 8-4):

1. **Minimum** – nalezená nejmenší hodnota účelové funkce v dané sérii, tj. nejmenší výsledná hodnota ze série optimalizačních experimentů s konkrétním nastavením parametrů optimalizačního algoritmu.
2. **Dolní kvartil** (1. kvartil, tj. 25 % percentil) - 25 % hodnot účelové funkce je nižších než hodnota dolního kvartilu v sérii.
3. **Medián** - 50 % hodnot účelové funkce je nižších než hodnota mediánu v sérii.
4. **Horní kvartil** (3. kvartil, tj. 75 % percentil) - 75 % hodnot účelové funkce je nižších než hodnota horního kvartilu v sérii.
5. **Maximum** – nalezená největší hodnota účelové funkce v dané sérii tj. největší hodnota ze všech optimalizačních experimentů s konkrétním nastavením optimalizačního algoritmu.

Pomocí krabicového grafu lze rychle vizuálně hodnotit kvalitu nalézaných řešení (prvků) v rámci jedné série, která představuje jednu krabice v grafu.



Obr. 8-4 Příklad výsledků jedné série (jedné krabice) pomocí charakteristik krabicového grafu na základě hodnot účelové funkce

Podstata využití krabicového grafu je ilustrována na příkladu simulačního modelu výrobní linky, na kterou byl aplikován algoritmus Evoluční strategie. Cílem bylo nalézt maximální hodnotu účelové funkce. V následující tabulce (viz Tabulka 8-2) jsou zobrazeny hodnoty charakteristik jedné série spočtené z výsledků optimalizačních experimentů, tedy jedné krabice v krabicovém grafu. Vertikální osa v grafu určuje hodnoty účelové funkce výsledků v sérii. Hodnota účelové funkce globální optima $(F(\mathbf{X}^*))$ v prohledávaném prostoru $(\tilde{X})$ je $F(\mathbf{X}^*) = 22032$. K ukončení optimalizačního experimentu dojde tehdy, pokud je nalezeno takové řešení, které dosahuje této hodnoty účelové funkce nebo je v rozsahu tolerované odchylky.

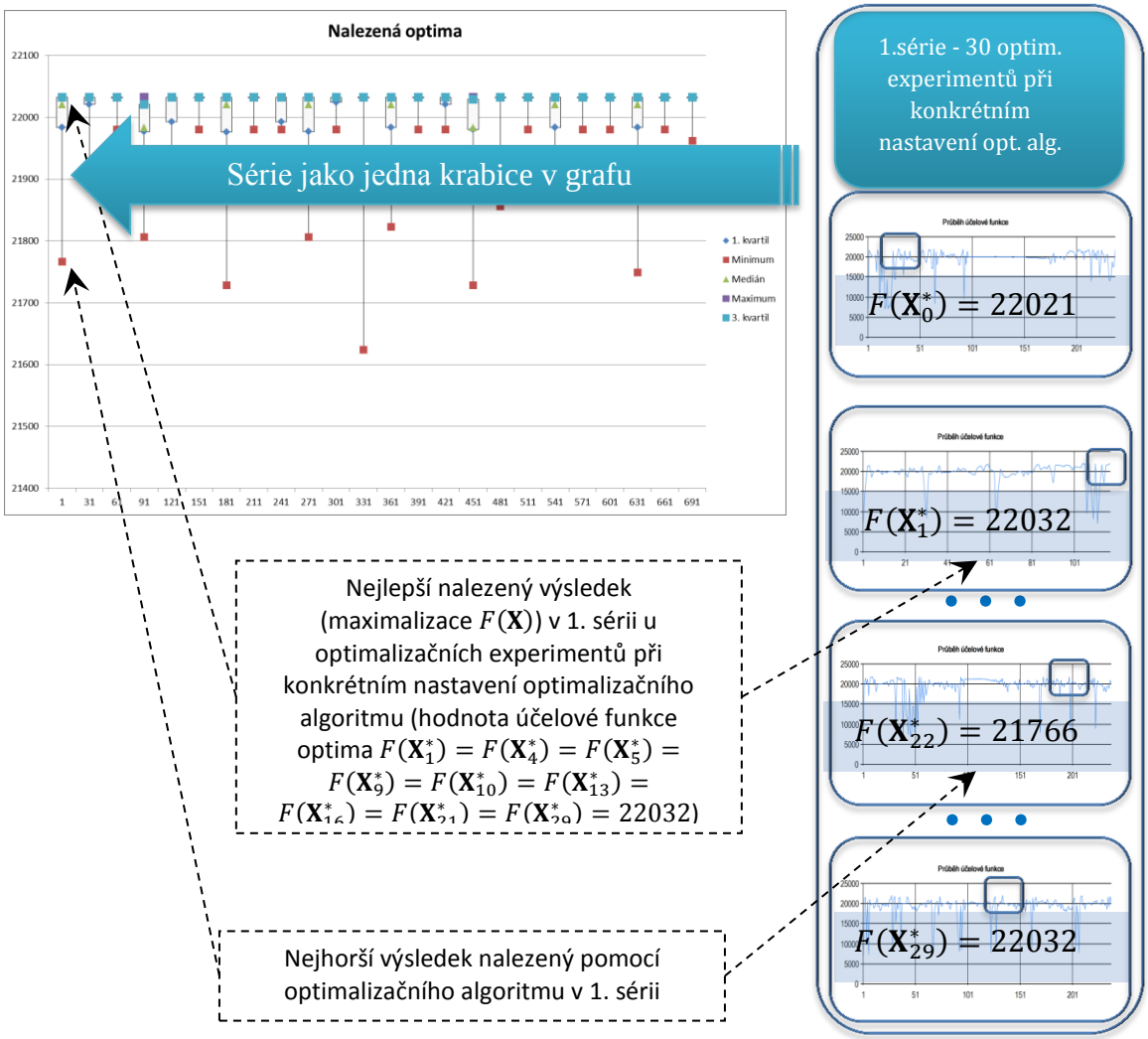
Charakteristika krabicového grafu	Označení charakteristiky	Hodnota charakteristiky
Minimum	Q_1	21766
Dolní kvartil	Q_2	21983
Medián	Q_3	22021
Horní kvartil	Q_4	22032
Maximum	Q_5	22032

Tabulka 8-2 Charakteristiky krabicového grafu vztahující se k výsledkům poskytnutých optimalizačním algoritmem Evoluční strategie v 1. sérii – tj. při určitém nastavení parametrů optimalizačního algoritmu – simulační model výrobní linky - maximalizace účelové funkce

Na následujícím obrázku (viz Obr. 8-5) je nastíněno chování optimalizačního algoritmu „Evoluční strategie“ při různých nastaveních parametrů algoritmu. V krabicovém grafu jsou zobrazeny rozsahy hodnot účelové funkce nalezených výsledků v jednotlivých sériích prováděných optimalizačním algoritmem. Platí tedy, že čím jsou jednotlivé charakteristiky určité série blíže k hodnotě účelové funkce globálního maxima (u simulačního modelu výrobní linky se jedná o maximalizaci hodnot účelové funkce), tím jsou kvalitnější. Je patrné, že jednotlivé krabice v grafu (identifikující jednotlivé série) jsou různorodé. Znamená to, že optimalizační algoritmus je

náchylný na nastavení vlastních parametrů – každá série obsahuje jiné nastavení parametrů optimalizačního algoritmu. Pro každou sérii ze získaných výsledků optimalizačních experimentů jsou pomocí softwarové aplikace simulační optimalizace v excelovském souboru vypočteny charakteristiky krabicových grafů a tyto grafy jsou následně sestrojeny. Pro hodnocení chování algoritmu jsou sestrojeny 3 typy krabicových grafů s názvy:

1. **„Nalezená optima“** – rozsah hodnot účelové funkce nejlepších nalezených výsledků (v nejlepším případě optim) ze všech optimalizačních experimentů provedených v rámci jedné série je zobrazen pomocí jedné krabice. Tato „krabice s vousy“ reprezentuje vypočtené kvartilové charakteristiky z těchto hodnot. Všechny optimalizační experimenty série proběhly s jedním konkrétním nastavením parametrů optimalizačního algoritmu. V krabicovém grafu jsou pomocí krabic vykresleny všechny provedené série (viz Obr. 8-5).



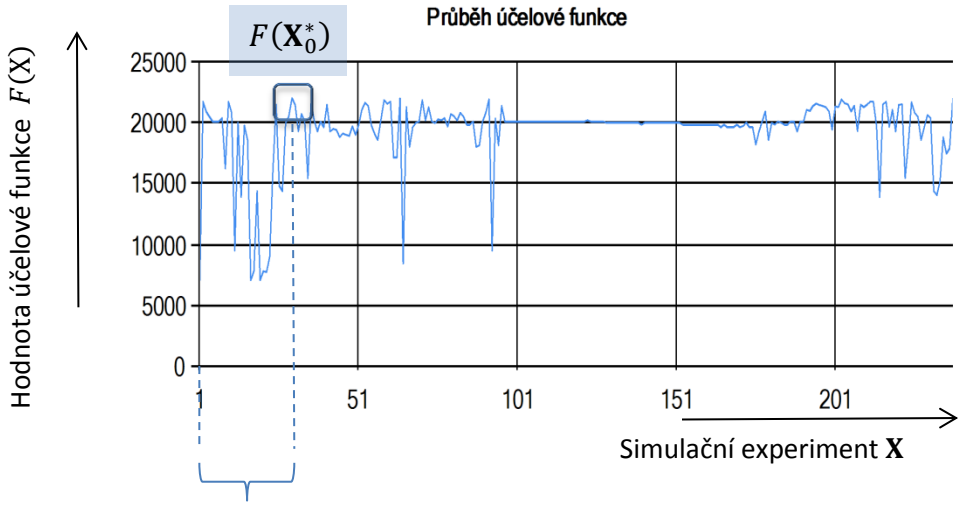
Obr. 8-5 Příklad výsledků v krabicovém grafu poskytnutých optimalizačním algoritmem „Evoluční strategie“ – jedna série (jedno konkrétní nastavení) je zobrazena jako jedna krabice v grafu – simulační model výrobní linky - maximalizace účelové funkce

kde:

- $\mathbf{X}_i^*$ … i-tý prvek množiny optim. Nalezený možný kandidát řešení optimalizačním algoritmem … $\mathbf{X}_i^* \in X^*, X^* \subseteq \tilde{X}, \forall i: 0 < i \leq m^*$.
- X^* … Množina optim.

- i … Index prvku množiny optim.
- $\tilde{X}$ … Prohledávaný prostor optimalizačním algoritmem.
- m^* … Počet prvků množiny optim (každý optimalizační experiment s konkrétním nastavením parametrů optimalizačního algoritmu poskytl pouze jedno řešení, tzn., že množina optim série obsahuje 30 prvků ... $m^* = 30$).
- $F(\mathbf{X})$ ... Hodnota účelové funkce.
- $F(\mathbf{X}_0^*)$ ... Hodnota účelové funkce nalezeného optima v 1. optimalizačním experimentu v sérii.

2. **„Rychlost nalezení optima“** – jedna krabice v krabicovém grafu reprezentuje vypočtené kvartilové charakteristiky z rozsahu hodnot počtu provedených simulačních experimentů, než došlo k vyhledání nejlepšího výsledku v optimalizačním experimentu. Pokud bylo provedeno 30 optimalizačních experimentů v rámci jedné série, kvartilové charakteristiky budou spočteny z 30 hodnot. V následujícím obrázku (viz Obr. 8-6) je uveden příklad prvního optimalizačního experimentu v konkrétní sérii. Z obrázku je patrné, že je sledováno, kolik bylo zapotřebí provést simulačních experimentů, než byl nalezen nejlepší výsledek - lokální extrém (lokální maximum) účelové funkce – hodnota účelové funkce tohoto prvku je označena jako $F(\mathbf{X}_0^*)$.



Počet provedených simulačních experimentů do nalezení nejlepšího prvku 1. optimalizačního experimentu

Obr. 8-6 Počet provedených simulačních experimentů do nalezení optima (nalezené řešení není globálním optimem) při 1. optimalizačním experimentu s konkrétním nastavením optimalizačního algoritmu v první sérii – maximalizace účelové funkce

3. „Hodnoty účelové funkce při experimentech“ – pomocí krabice v krabicovém grafu jsou zobrazeny kvartilové charakteristiky spočtené z hodnot účelové funkce ze **všech** provedených simulačních experimentů (tj. ze všech simulačních experimentů všech optimalizačních experimentů) v rámci jedné série s konkrétním nastavením optimalizačního algoritmu.

9 Simulační modely

Pro testování jednotlivých optimalizačních algoritmů a jejich chování u různých typů účelové funkce, byly vybrány následující funkce, které jsou běžně využívány při testování optimalizačních algoritmů ve spojité optimalizaci. Tyto funkce představující účelovou funkci, která bude optimalizována. Hodnota funkce může však být vypočtena pouze v bodech – prvcích o souřadnicích rozhodovacích proměnných, které jsou určeny krokem definovaným pro jednotlivou rozhodovací proměnnou. Nelze tedy provést derivaci v bodě.

Podrobný popis následujících simulačních modelů včetně specifikace rozhodovacích proměnných, prohledávaného prostoru, účelové funkce a jejího průběhu, globálního minima a maxima v prohledávaném prostoru a definovaných kritérií ukončení jsou uvedeny v příloze (viz **příloha** – kapitola 5 Simulační modely).

9.1 „De Jong“

Simulační model charakterizuje De Jongovu funkci - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.1 Simulační model „De Jong“). Tato konvexní funkce představuje *n*-rozměrnou parabolu. Funkce je snadnou úlohou pro optimalizační metody. Cílem je minimalizovat účelovou funkci v rámci definovaného prohledávaného prostoru. Předpis účelové funkce je specifikován: [14]

$$F(\mathbf{X}) = \sum_{j=1}^{n} {x_j}^2 \tag{9.1}$$

kde:

- $F(\mathbf{X})$ … Účelová funkce.
- n … Počet dimenzí prostoru.
- j … Index rozhodovací proměnné v prohledávaném prostoru.
- x_j … Hodnota souřadnice bodu – prvku rozhodovací proměnné.

9.2 „Rosenbrock“

Simulační model charakterizuje Rosenbrockovu (též označovanou „banánovou“, „sedlovou“ nebo druhou De Jongovu) funkci - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.2 Simulační model „Rosenbrock“). Tato nekonvexní funkce představuje dlouhé, úzké, parabolické ploché údolí. Funkce je středně obtížnou úlohou pro optimalizační metody. Cílem je minimalizovat účelovou funkci v rámci definovaného prohledávaného prostoru. Předpis účelové funkce je specifikován: [14]

$$F(\mathbf{X}) = \sum_{j=1}^{n-1} 100 \cdot \left({x_j}^2 - x_{j+1}\right)^2 + \left(1 - x_j\right)^2 \tag{9.2}$$

9.3 „Michalewicz“

Simulační model charakterizuje Michalewiczovu funkci - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.3 Simulační model „Michalewicz“). Tato funkce představuje n-rozměrnou multimodální účelovou funkci (obsahující $n!$ lokálních optim). Funkce je těžkou úlohou pro optimalizační metody. Cílem je minimalizovat účelovou funkci v rámci definovaného prohledávaného prostoru. Předpis účelové funkce je specifikován: [32]

$$F(\mathbf{X}) = -\sum_{j=1}^{n} \sin(x_j) \cdot \left(sin\left(\frac{j \cdot {x_j}^2}{\pi} \right) \right)^{2 \cdot m} \tag{9.3}$$

kde:

- m … Parametr ovlivňující příkrost údolí a hran. Pro naši účelovou funkci je stanoveno $m = 5$.

Pro účelovou funkci bylo stanoveno $m = 5$. Parametr m stanovuje příkrost údolí a hran. Větší m vede k těžšímu prohledávání účelové funkce. Pro velké m funkce vypadá jako jehla v kupce sena (hodnoty účelové funkce bodů mimo úzké vrcholky poskytují velmi málo informací o pozici globálního optima).

9.4 „Ackley“

Simulační model charakterizuje Ackleyho funkci - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.4 Simulační model „Ackley“). Tato funkce je multimodální – obsahuje mnoho vrcholů. Funkce je dobrým testem předčasné konvergence („zaseknutí“ v lokálním optimu) pro optimalizační metodu. Cílem je minimalizovat účelovou funkci v rámci definovaného prohledávaného prostoru. Předpis účelové funkce je specifikován: [14]

$$F(\mathbf{X}) = -20 \cdot \exp\left(-0.02 \cdot \sqrt{\frac{1}{n} \cdot \sum_{j=1}^{n} {x_j}^2} \right) - \exp\left(\frac{1}{n} \cdot \sum_{j=1}^{n} \cos 2\pi x_j \right) + 20 + \exp(1) \tag{9.4}$$

kde:

- $\exp(1) = e^1$ … Vrátí *e* umocněné na hodnotu argumentu – číslo v závorce.

Dále byly pro testování optimalizačních algoritmů použity následující diskrétní simulační modely vytvořené v simulačním nástroji ARENA.

9.5 „Doprava“

Simulační model z praxe zachycuje celkovou interní logistiku ve výrobní hale a skladu - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.5 Simulační model „Doprava“). Doprava je zajišťována pomocí vláčků, ke kterým se zapojují vagóny. V modelu jsou definovány jednotlivé dopravní cesty. Při navážení materiálů ze skladu vláček dle potřeby zaváží předmontážní pracoviště a všechny montážní linky. Trasa, kterou vláček projede, je odvozena od cílových míst přepravovaného materiálu. Některé dílce jsou přepravovány z předvýroby do skladu.

Existují následující skupiny vláčků, které zajišťují pouze dopravu:

1. Doprava malých dílců pro montážní linky a pro předmontáže v předvýrobě.
2. Doprava velkých dílců pro montážní linky.
3. Odvoz hotových výrobků od montážních linek a navážení obalových materiálů.

Účelová funkce odráží průměrné využití vláčků s vagóny, které slouží pro transport malých a velkých dílců a hotových výrobků a celkové průměrné využití všech montážních linek. Průměrné využití všech montážních linek je v účelové funkci nadřazeno průměrnému využití u všech typů vláčků pomocí koeficientů. Cílem je nalézt vhodnou konfiguraci počtu vláčků, které zaváží komponenty do výroby, montáže a odváží hotové výrobky.

9.6 „Penalizace“

Simulační model představuje výrobní dílnu, kde jsou vyráběny dva druhy výrobků - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.6 Simulační model „Penalizace“). Výrobní dílna obsahuje 8 technologicky orientovaných pracovišť. Každé pracoviště disponuje stanoveným počtem zdrojů. Tento počet byl

určen na základě statického odhadu počtu zdrojů podle stanoveného objemu produkce a definovaného času obrábění na pracovišti.

Každý výrobek má svůj technologický postup – čas obrábění a operace, která má být provedena na daném typu výrobku. Cílem je vyrobit 100 kusů prvního výrobku a 200 kusů druhého výrobku za dobu 6350 minut. Za nevyrobení prvního výrobku je stanovena penalizace 200 Kč a za druhý je penalizace stanovena částkou 300 Kč. Pokud výrobek přesáhne stanovenou průběžnou dobu výrobu, bude penalizován. Penalizace nastává i v případě, že výrobek je předčasně vyroben. V rámci simulačního modelu budou měněny vstupy obou výrobků na dílnu. Účelová funkce bude zachycovat výši penalizace při nesplnění v požadovaném termínu dodání. Je snahou docílit minimální výše penalizace.

9.7 „VyrobniLinka“

Simulační model z praxe charakterizuje výrobní linku pro dva různé výrobní postupy včetně relativní chybovosti na jednotlivých pracovištích - detailní popis simulačního modelu je uveden v příloze (viz **příloha** – kapitola 5.7 Simulační model „VyrobniLinka“). Doprava ve výrobě je uskutečněna pomocí pásového dopravníku. Linka se skládá z 11 pracovišť. Jsou definovány jednotlivé časy zpracování a jejich odchylky. Na konci výrobní linky jsou výrobky tříděny podle toho, zda jsou vadné či nikoliv. Pokud vznikl na jakémkoliv pracovišti zmetek, tento zmetek již na následujících pracovištích není dále zpracován. Cílem optimalizace je nalezení vhodných hodnot počtu upínacích přípravků inicializujících přechod pracovníka mezi pracovišti a dále počtu upínacích přípravků v oběhu.

10 Experimentování

Pro každý simulační model byla nejprve provedena specifikace jednotlivých atributů nutných pro provedení optimalizačních experimentů. Jednotlivá kritéria definovaná pro simulační modely jsou uvedena v příloze (viz **příloha** – kapitola 5 Simulační modely).

- Specifikace rozhodovacích proměnných v prohledávaném prostoru.
- Specifikace prohledávaného prostoru - hranice prohledávaného prostoru u testovaných funkcí byly stanoveny tak, aby prohledávaný prostor obsahoval globální optimum na celém prostoru všech řešení a povrch účelové funkce byl typický pro průběh účelové funkce.
- Specifikace účelové funkce.
- Zmapování průběhu účelové funkce v rámci prohledávaného prostoru u simulačního modelu, včetně uložení výsledků provedeného prořezu účelové funkce, v rámci specifikovaného prohledávaného prostoru, do znalostní databáze ve formě *.xml. Díky tomu při novém optimalizačním experimentu simulační experimenty nemusí být znovu prováděny, ale načítají se z importované znalostní databáze. Tím se značně sníží výpočetní čas optimalizačních experimentů.
- Identifikace globálního minima a maxima účelové funkce v prohledávaném prostoru.
- Specifikace kritérií ukončení.
- Specifikace času potřebného k proběhnutí simulačního experimentu (včetně popsání parametrů počítače, na kterém byly experimenty prováděny).
- U simulačních modelů vytvořených v simulačním nástroji ARENA byly specifikovány parametry simulačního běhu na simulačním modelu (simulační doba v simulačním modelu, doba náběhu simulačního modelu, pracovní doba).

10.1 Série – formulace podmínek

U všech prováděných optimalizační experimentů (včetně sérií) se simulačním modelem byly pomocí optimalizačního algoritmu zachovány stejné podmínky. Stejnými podmínkami se v tomto případě rozumí:

- Stejné výchozí body (pokud nebylo zvoleno náhodné generování počátečních přípustných řešení). Výchozím bodem optimalizačních experimentů (kromě modelu „Doprava“) je bod opačného globálního extrému (v případě minimalizace účelové funkce je to globální maximum v prohledávaném prostoru).
- Stejný počet optimalizačních experimentů v každé sérii.
- Stejná kritéria ukončení – disjunktivní podmínky typu:
    - Definování počtu iterací, po kterých bude optimalizační experiment s konkrétním nastavením ukončen.
    - Dosažení globálního optima.
    - Dosažení lokálního extrému v definované toleranci hodnoty účelové funkce od hodnoty účelové funkce globálního optima. V případě Diferenciální evoluce a Downhill Simplexu bylo kritérium ukončení nastaveno na hodnotu 500 iterací nebo pokud při optimalizaci nedocházelo k zlepšení suboptima v poměru 0.003 (viz kapitola 8.2.2). U těchto optimalizačních algoritmů, díky jejich podstatě, docházelo k oscilaci nebo ke kopírování stejných nalezených řešení do dalších populací a hodnota účelové funkce nejlepšího nalezeného řešení zůstávala konstantní.

Za účelem zkoumání vlivu různého nastavení parametrů optimalizačních algoritmů byla v aplikaci implementována experimentální základna s možností nastavení dolní a horní meze pro každý parametr optimalizačního algoritmu. Po nastavení přírůstku (kroku) parametru optimalizačního algoritmu v rozsahu dolní a horní meze je parametr měněn o tento přírůstek a následně jsou prováděny optimalizační experimenty pro konkrétní nastavení (lze také nastavit počet experimentů v sérii pro konkrétní nastavení).

Aby bylo možné určit vhodné nastavení parametrů optimalizačního algoritmu, bylo zapotřebí provést u všech optimalizačních algoritmů velký počet sérií = konkrétních nastavení parametrů optimalizačního algoritmu na všech simulačních modelech. Na všech simulačních modelech pro všechny optimalizační algoritmy bylo provedeno celkem **318 430** různých nastavení parametrů optimalizačních algoritmů (= 45490 sérií × 7 simulačních modelů). Počet sérií je závislý na počtu parametrů algoritmu. Rozvržení jednotlivých sérií je vidět v následující tabulce (viz Tabulka 10-1).

Optimalizační algoritmus	Parametr algoritmu	Krok	Meze		Celkový počet sérií
			Dolní mez	Horní mez	
Random Search	Generovat stejné prvky (Ano/Ne)	1	0	1	2
Hill Climbing	Rozptyl	2	1	27	392
	Velikost populace	2	1	27	
	První populace náhodně (Ano/Ne)	1	0	1	
Tabu Search	Rozptyl	2	1	27	3920
	Velikost populace	2	1	27	
	Tabu Length	10	10	50	
	První populace náhodně (Ano/Ne)	1	0	1	
	Optimalizace podle poslední populace (Ano/Ne)	1	0	1	
Simulated Annealing	Rozptyl	2	1	13	7056
	Měnit jen jeden parametr (Ano/Ne)	1	0	1	
	Beta	0.14	0.1	0.94	
	Minimální teplota	0.04	0.01	0.33	
	Generovat dle rozptylu (Ano/Ne)	1	0	1	
	Snižovat teplotu jen při přijetí horšího řešení (Ano/Ne)	1	0	1	
	První populace náhodně (Ano/Ne)	1	0	1	
Downhill Simplex	Reflexe	0.2	0	2	24200
	Expanze	0.2	0	2	
	Kontrakce	0.11	0	0.99	
	Redukce	0.11	0	0.99	
	První populace náhodně (Ano/Ne)	1	0	1	
Local Search	Rozptyl	1	1	30	60
	První populace náhodně (Ano/Ne)	1	0	1	
Differential Evolution	Velikost populace	2	6	30	2600
	Koef_F	0.11	0	0.99	
	Koef_C	0.11	0	0.99	
	První populace náhodně (Ano/Ne)	1	0	1	
Evolution Strategy	Velikost populace	2	1	21	7260
	Mmp	2	1	21	
	q	5	1	21	
	k	4	1	21	
	První populace náhodně (Ano/Ne)	1	0	1	
				∑ provedených sérií	45490

Tabulka 10-1 Provedené série na simulačních modelech

Pro každou konkrétní sérii jsou získané výsledky všech provedených simulačních experimentů (hodnoty účelové funkce a jejích parametrů) průběžně exportovány do excelovského souboru (*.xlsx). V excelovském souboru (*.xlsx) jsou dále uloženy:

- Jméno zvoleného optimalizačního algoritmu.
- Nastavení parametrů zvoleného optimalizačního algoritmu (včetně dolní meze – min, horní meze - max, kroku parametru) – každý simulační experiment má vlastní identifikátor – číslo nastavení.
- Počet opakování konkrétního nastavení v každé sérii.
- Nastavení kritérií ukončení včetně informace, které kritérium ukončení bylo splněno, a tudíž byl zastaven běh optimalizačního algoritmu.
- Jméno simulačního modelu – pokud se jedná o vlastní sestavenou účelovou funkci pomocí aplikace simulační optimalizace, je zobrazen její předpis.
- Nejlepší nalezená hodnota účelové funkce.
- Hodnoty účelové funkce všech experimentů zvoleného algoritmu.
- Hodnoty rozhodovacích proměnných (vstupní parametry simulačního modelu) u každého simulačního experimentu (včetně počátečního přípustného řešení).

10.2 Časová náročnost a vzdálené řízení experimentů

Za účelem testování chování optimalizačních algoritmů na simulačních modelech bylo provedeno cca **4 112 000 000** simulačních experimentů – objem dat cca **250 GB**. Časová náročnost provedení všech simulačních experimentů na jednom počítači by trvala zhruba **9760** roků (odvozeno od potřebného času na proběhnutí simulačního experimentu na používaném PC pro výpočty). Proto byl proveden systematický prořez všech možných variant řešení v prohledávaném prostoru a výsledky prořezu byly uloženy do znalostních databází simulačních modelů. Díky využití znalostní databáze jednotlivé varianty možného řešení nemusely být znovu počítány pomocí simulačního experimentu, ale byly vyhledány v databázi výsledků. Tím časová náročnost klesla na cca **2.35** roků. Aplikací tohoto přístupu došlo k urychlení zhruba **4000**-krát. Vzhledem ke stále velké časové náročnosti výpočtu bylo využito distribuce na více počítačů (12 PC). Tím časová náročnost klesla na cca **0.2** roků. Pro vzdálené řízení simulačních experimentů byl využit program, který běžel jako služba na každém počítači. Program dosahoval vyšší rychlosti při stahování souborů s výsledky optimalizačních experimentů oproti „Připojení ke vzdálené ploše“, která je zahrnuta v operačním systému Windows XP.

11 Metodika hodnocení optimalizačních algoritmů podle různých kritérií

Automaticky sestrojené krabicové grafy sice na počátku experimentování poskytovaly rychlý vizuální přehled o kvalitě provedených sérií z různých pohledů, ale problém nastal tehdy, když bylo provedeno velké množství sérií, tj. v krabicových grafech je vykreslen velký počet krabic. Objem dat výsledků realizovaných simulačních experimentů byl zhruba **250 GB**. Byla proto následně navržena metodika hodnocení optimalizačních experimentů v sérii, která umožňuje zachytit různé pohledy na chování optimalizačního algoritmu.

Metodika pro hodnocení zahrnuje pět funkcí, které využívají nejen vypočtených charakteristik krabicových grafů (byly implementovány do prostředí aplikace simulační optimalizace), ale využívá také data z jednotlivých optimalizačních experimentů. Funkce zachycují pět základních **kritérií** hodnocení chování každého optimalizačního algoritmu, kterými jsou:

1. Počet nalezených optim nebo hodnot blízkých optimu - f_1.
2. Rozdíl nalezeného „extrému“ od optima - f_2.
3. Vzdálenost kvartilů - f_3.
4. Rychlost nalezení optima (popřípadě lokálního „extrému“ – nejlepšího nalezeného řešení) - f_4.
5. Konvergence - f_5.

Každá funkce je normalizována na hodnotu v rozsahu [0,1], a **její co nejmenší hodnota (nejlépe blížící se k nule) představuje lepší hodnocení.** Ohodnocující funkce jsou dále využity při hledání vhodného nastavení a hodnocení optimalizačního algoritmu. Cílem je minimalizace všech hodnot (první z nich mají větší důležitost).

V následujících kapitolách budou popsány principy uplatněné metodiky ohodnocení jednotlivých sérií vybraných optimalizačních algoritmů na všech simulačních modelech podle předchozích specifikovaných pěti kritérií.

11.1 Kritérium - f_1 - Počet nalezených optim nebo hodnot blízkých optimu

V této kapitole je popsán princip ohodnocení jednotlivých optimalizačních experimentů na základě nalezení globálního optima, tj., kvantifikace experimentálních výsledků pomocí f_1. U všech sedmi simulačních modelů byl proveden prořez celého prohledávaného prostoru, díky němuž mohlo být určeno globální optimum účelové funkce v tomto prostoru (referenční hodnota pro vyhodnocování). Na základě rozsahu jednotlivých charakteristik krabicového grafu „Nalezená optima“, který byl automaticky generován aplikací simulační optimalizace, lze úspěšnost u každé jednotlivé série velice rychle vizuálně ohodnotit. Toto vizuální hodnocení přestává být přehledné při velkém počtu sérií, tj. při velkém množství konkrétních nastavení parametrů optimalizačního algoritmu. Z tohoto důvodu je specifikována následující funkce (viz Algoritmus 11-1) ohodnocující kvalitu všech nalezených optim dané série (popřípadě lokálních extrémů blízkých optimu). Výstupem této funkce je normalizovaná hodnota v rozsahu $f_1 \in [0,1]$. Cílem je **minimalizace** této hodnoty, to znamená, že pro hodnocenou sérii může v nejlepším případě být $f_1 = 0$. V tomto případě je série nazývána jako **absolutně úspěšná** série. Tento ideální výsledek nastává tehdy, když u všech optimalizačních experimentů v sérii s konkrétním nastavením optimalizačního algoritmu bylo vždy nalezeno optimum nebo lokální extrém, který svojí hodnotou účelové funkce je velmi blízký optimu (v rozsahu definované tolerance).

$f_1 \leftarrow \text{AssignFitnessFindingOptimum}(X^*, \mathbf{X}^*, \varepsilon)$	
Funkce, jejímž výstupem je skalární normovaná hodnota v rozsahu $f_1 \in [0,1]$. Tato hodnota představuje ohodnocení neúspěšnosti všech nalezených řešení optimalizačním algoritmem v konkrétní sérii – minimalizace této hodnoty	
Vstup:	X^*: Seznam nalezených optim v konkrétní sérii – každý optimalizační experiment má své nalezené optimum
Vstup:	$\mathbf{X}^*$: Globální optimum $\mathbf{X}^*$ v prohledávaném prostoru

Vstup:	ε:	Tolerovaná odchylka od hodnoty účelové funkce globálního optima
Data:	$F(\mathbf{X})$:	Účelová funkce
Data:	n_{Succ}:	Čítač úspěchů
Výstup:	f_1:	Skalární normovaná hodnota ohodnocení kvality nalezených řešení

1 **begin**
2 $n_{Succ} \longleftarrow 0$;
3 **for** $i \longleftarrow 0$ **to** $\mathrm{Length}(X^*) - 1$ **do**
4 **if** $|F(X^*[i]) - F(\mathbf{X}^*)| \leq \varepsilon$ **then**
5 $n_{Succ} \longleftarrow n_{Succ} + 1$;
(*bylo nalezeno optimum popřípadě přijatelné řešení, které je v mezích tolerované odchylky*)
6 **result** $\longleftarrow \frac{\mathrm{Length}(X^*) - n_{Succ}}{\mathrm{Length}(X^*)}$; //normování (% podíl neúspěšných sérií)
7 **end**;

Algoritmus 11-1 AssignFitnessFindingOptimum

Ohodnocení výsledků sérií podle f_1 – ohodnocení počtu nalezených optim nebo hodnot blízkých optimu

Na základě předchozího popsaného principu ohodnocení úspěšnosti nalezení optima bylo provedeno ohodnocení všech sérií všech vybraných optimalizačních algoritmů na sedmi simulačních modelech:

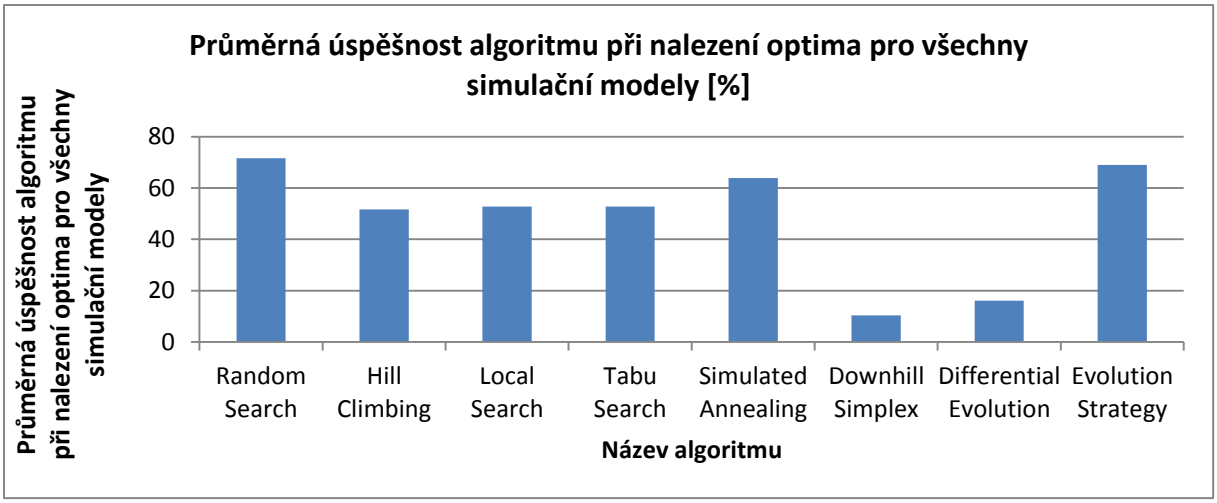
- „De Jong“.
- „Rosenbrock“.
- „Michalewicz“.
- „Ackley“.
- „Doprava“
- „Penalizace“.
- „Linka“.

Pro ohodnocení byl použit předcházející algoritmus (viz Algoritmus 11-1). Na základě vypočtených hodnot kritéria f_1 byl sestrojen graf **průměrné úspěšnosti** všech zvolených optimalizačních algoritmů při nalezení optima na simulačních modelech. Graf poskytuje celkový přehled o všech výsledcích vypočtené úspěšnosti (pro všechny série při všech nastaveních parametrů algoritmu) u jednotlivých algoritmů (viz Obr. 11-1). Hodnoty procentuální průměrné úspěšnosti byly spočteny podle vztahu:

$$\mathrm{f}_{\mathrm{Avg}} = \left(1 - \frac{\sum_{i=1}^{s} \mathrm{f}_{1_i}}{s}\right) \cdot 100 \tag{11.1}$$

kde:

- $\mathrm{f}_{\mathrm{Avg}}$ … Procentuální průměrná úspěšnost optimalizačního algoritmu na základě úspěšnosti nalezení optima.
- i ... Index jednotlivé série.
- s ... Celkový počet provedených sérií.
- f_{1_i} ... Hodnota kritéria vyjadřujícího neúspěšnost při nalezení optima pro i-tou sérii (minimalizace této hodnoty - nejlepší dosažená hodnota může nabývat $\mathrm{f}_1 = 0$).

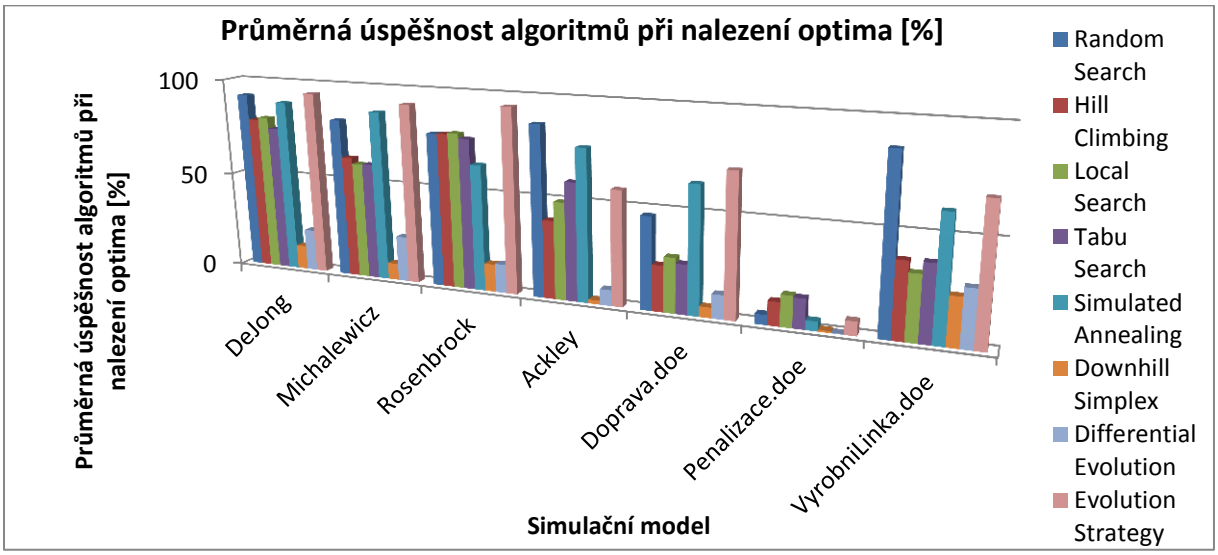


Obr. 11-1 Průměrná úspěšnost algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu

Ze sloupcového grafu (viz Obr. 11-1) vyplývá, že nejúspěšnějším algoritmem bylo Náhodné prohledávání - Random Search. Tento algoritmus je ale velmi závislý na počtu prováděných simulačních experimentů a díky tomu, že u kritéria ukončení byl specifikován maximální počet provedených simulačních experimentů na hodnotu rovnou počtu všech možných řešení v prohledávaném prostoru, algoritmus dosahoval „nadhodnocené“ úspěšnosti. Algoritmus je využitelný zejména v případech, kdy nemáme žádnou představu o průběhu účelové funkce a jsme schopni provést větší počet simulačních experimentů (metoda Monte Carlo).

Dalším úspěšným algoritmem je Evoluční strategie - Evolution Strategy. Podobně se umístil optimalizační algoritmus Simulovaného žíhání – Simulated Annealing. Tento algoritmus patří do oblasti pseudogradientních metod a vyniká díky své vlastnosti přijímat i horší řešení, ale s postupně se snižující pravděpodobností.

Předchozí graf (viz Obr. 11-1) je koncentrovaným (souhrnným) pohledem na průměrnou úspěšnost optimalizačního algoritmu při nalezení optima na všech sedmi simulačních modelech. Při dále uvedeném rozkladu této, dá se říci univerzální, průměrné úspěšnosti optimalizačního algoritmu, na průměrnou úspěšnost při nalezení optima pro každý simulační model (viz Obr. 11-2) je patrné, že optimalizační algoritmy dosahují rozdílné průměrné úspěšnosti v závislosti na průběhu účelové funkce u jednotlivých simulačních modelů.



Obr. 11-2 Průměrná úspěšnost algoritmů při nalezení optima pro jednotlivé simulační modely a všechny nastavené parametry algoritmu

V grafu výsledků průměrné úspěšnosti jednotlivých algoritmů u jednotlivých simulačních modelů (viz Obr. 11-2) je patrné a není překvapivé, že algoritmy jsou závislé na průběhu účelové funkce. U jednoduchých průběhů účelové funkce (jakými jsou De Jong, Rosenbrock) je průměrná úspěšnost algoritmů vysoká na rozdíl od velmi členitého povrchu účelové funkce simulačního modelu „Penalizace“.

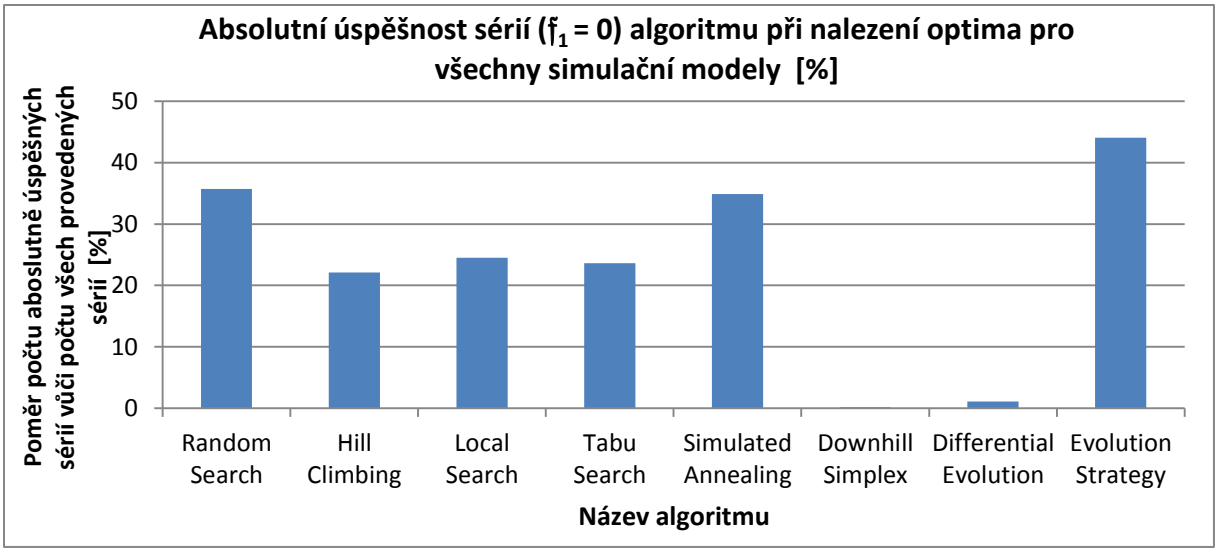
Je třeba si uvědomit, že v předchozích grafech jsou zahrnuty všechny série, tedy i naprosto **neúspěšné** série, jejichž neúspěch byl ovlivněn **špatným** nastavením parametrů optimalizačního algoritmu. Z tohoto důvodu byl specifikován následující algoritmus ohodnocení jednotlivých sérií na základě kritéria f_1 (viz Algoritmus 11-2) pouze pro takové série, jejichž vypočtená hodnota f_1 na základě „AssignFitnessFindingOptimum“ bude lepší nebo stejná, než specifikovaná hodnota vstupního kritéria „$crit$“.

$f_{1succ} \leftarrow \text{SeriesSuccess}(S, crit, X^*, \mathbf{X}^*, \varepsilon)$		
Funkce, jejímž výstupem je hodnota procentuálního poměru počtu úspěšných sérií – nastavení - algoritmu (na základě stanoveného kritéria) vůči všem provedeným sériím. Úspěšná série zde musí splňovat podmínku, že hodnota první ohodnocující funkce „AssignFitnessFindingOptimum“ musí být rovna hodnotě vstupního kritéria.		
Vstup:	S:	Seznam jednotlivých sérií
Vstup:	$crit$:	Hodnota kritéria
Vstup:	X^*:	Seznam nalezených optim v konkrétní sérii
Vstup:	$\mathbf{X}^*$:	Globální optimum $\mathbf{X}^*$ v prohledávaném prostoru
Vstup:	ε:	Tolerovaná odchylka od hodnoty účelové funkce globálního optima
Data:	f_1:	Skalární normovaná hodnota ohodnocení kvality nalezených řešení
Data:	n_{Succ}:	Čítač úspěchů
Výstup:	f_{1succ}:	Procentuální úspěšnost sérií

```
1   begin
2     n_Succ ← 0;
3     for i ← 0 to Length(S) − 1 do begin
4         f1 ← AssignFitnessFindingOptimum(X*, X*, ε)
    // hodnota první ohodnocující funkce pro konkrétní sérii
5         if f1 ≤ crit then
6             n_Succ ← n_Succ + 1 ;
    (*hodnota první ohodnocující funkce konkrétní série je stejná nebo lepší než hodnota kritéria*)
7         end;
8     result ← (n_Succ / Length(S)) * 100;        //převod na procentuální úspěšnost
9   end;
```

Algoritmus 11-2 SeriesSuccess

Jak již bylo řečeno, pro omezení vlivu špatného nastavení parametrů optimalizačního algoritmu, byly do výběru zahrnuty pouze takové série, jejichž hodnota byla $f_1 = 0$. Hodnota kritéria je stanovena záměrně velmi tvrdě (požadavek 100% úspěšnosti jednotlivé série při nalezení optima), protože algoritmům u malých prohledávaných prostorů bylo umožněno zkoumat celý prostor. Z výsledků byl sestrojen následující přehledný graf, který vyjadřuje poměr počtu absolutně úspěšných sérií podle f_1 (nalezení globálního optima nebo prvků, jejichž hodnota účelové funkce je v toleranci od hodnoty účelové funkce globálního optima) vůči počtu všech provedených sérií (viz Obr. 11-3).



Obr. 11-3 Absolutní úspěšnost sérií algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu

Při pohledu na předchozí graf (viz Obr. 11-3) je patrné, že nejúspěšnějším algoritmem s nejvyšším počtem absolutně úspěšných sérií, které vždy nalezly globální optimum, je algoritmus Evolution Strategy. Na výsledcích v grafu je také patrné, že absolutní úspěšnosti algoritmů Random Search a Simulated Annealing jsou téměř totožné. Samozřejmě platí, že hodnoty v grafu jsou ovlivněny počtem provedených sérií, tudíž výsledky Random Search by měly být brány s jistou opatrností.

Z grafu se dá usuzovat, že Evolution Strategy, Random Search a Simulated Annealing, by mohly být v jisté míře univerzální pro různé typy účelových funkcí. Špatných výsledků průměrné úspěšnosti dosahovaly Downhill Simplex – též někdy označovaná jako Simplexová metoda, a Diferenciální evoluce – Differential Evolution. Problém algoritmu Downhill Simplex spočívá v jeho podstatě, kdy je využíván bod reflexe, který je vypočten na základě těžiště všech bodů v simplexu. Aby bylo možné určit hodnotu účelové funkce v tomto bodu (prvku), bylo provedeno zaokrouhlení souřadnic směrem k nejbližšímu možnému bodu, určeného krokem u jednotlivé rozhodovací proměnné. Díky tomu dochází k mírnému ale vlivnému (v závislosti na velikosti rastru prohledávaného prostoru) odchýlení původního směru.

Pro otestování vlivu zaokrouhlování souřadnic na chování optimalizačního algoritmu Downhill Simplex byly navrženy následující série (viz Tabulka 11-1). V prohledávaném prostoru byla definována menší velikost kroku ($VelikostKroku = 1 \cdot 10^{-5}$), což by mělo vést k menšímu odchýlení optimalizačního algoritmu od původního směru, díky menšímu zaokrouhlování souřadnic rozhodovacích proměnných u vygenerovaného prvku. Takové experimentování lze provádět pouze na testovacích **funkcích**, kde lze nastavit **libovolnou** velikost kroku, na **rozdíl od diskrétních simulačních modelů**. Pro testování byl vybrán simulační model „DeJong“.

Zmenšením kroku $VelikostKroku = 1 \cdot 10^{-5}$ v prohledávaném prostoru, oproti předchozí testované hodnotě $VelikostKroku = 0.1$ (viz Tabulka 10-1), se také zvětšil počet prvků v prohledávaném prostoru z předchozích 1681 možných řešení na $1.6 \cdot 10^{11}$. Souřadnice rozhodovacích proměnných globálního maxima a globálního minima jsou stejné jako v předchozím případě, kdy byla $VelikostKroku = 0.1$ (viz **příloha** – kapitola 5.1 Simulační model „De Jong“). U nově provedených sérií byla zachována stejná kritéria ukončení jako v předchozím případě (viz **příloha** - Simulační model „De Jong“).

Parametr algoritmu	Krok	Meze		Série – počet opakování
		Dolní mez	Horní mez	
Reflexe	0.4	0	2	5
Expanze	0.4	0	2	
Kontrakce	0.33	0	0.99	
Redukce	0.33	0	0.99	
Reflexe	1	0	0	

Tabulka 11-1 Provedené série optimalizačního algoritmu Downhill Simplex s $VelikostKroku = 1 \cdot 10^{-5}$ na simulačním modelu „DeJong“

Na základě výsledků provedené analýzy byl potvrzen předpoklad, že pokud bude v prohledávaném prostoru nastavena menší velikost kroku, optimalizační algoritmus dosáhne větší efektivity při hledání optima. Průměrná úspěšnost optimalizačního algoritmu při nalezení optima u simulačního modelu „De Jong“ při nastavení $VelikostKroku = 1 \cdot 10^{-5}$ byla **34.7** % oproti průměrné úspěšnosti optimalizačního algoritmu při nalezení optima při nastavení $VelikostKroku = 0.1$, která činila pouhých **12.45** %.

Problém zaokrouhlování může být řešen modifikací algoritmu. Modifikaci bude třeba odvodit a ověřit pomocí aproximace hodnot účelové funkce v nejbližších bodech. Tím však stoupne operační náročnost výpočtu.

U algoritmu Differential Evolution dochází k tomu, že díky malému prohledávanému prostoru se do výběru dostávají identičtí jedinci. Tím je potlačen princip diverzity a algoritmus je zastaven na základě definovaného poměru zlepšení suboptima, který je součástí kritéria ukončení.

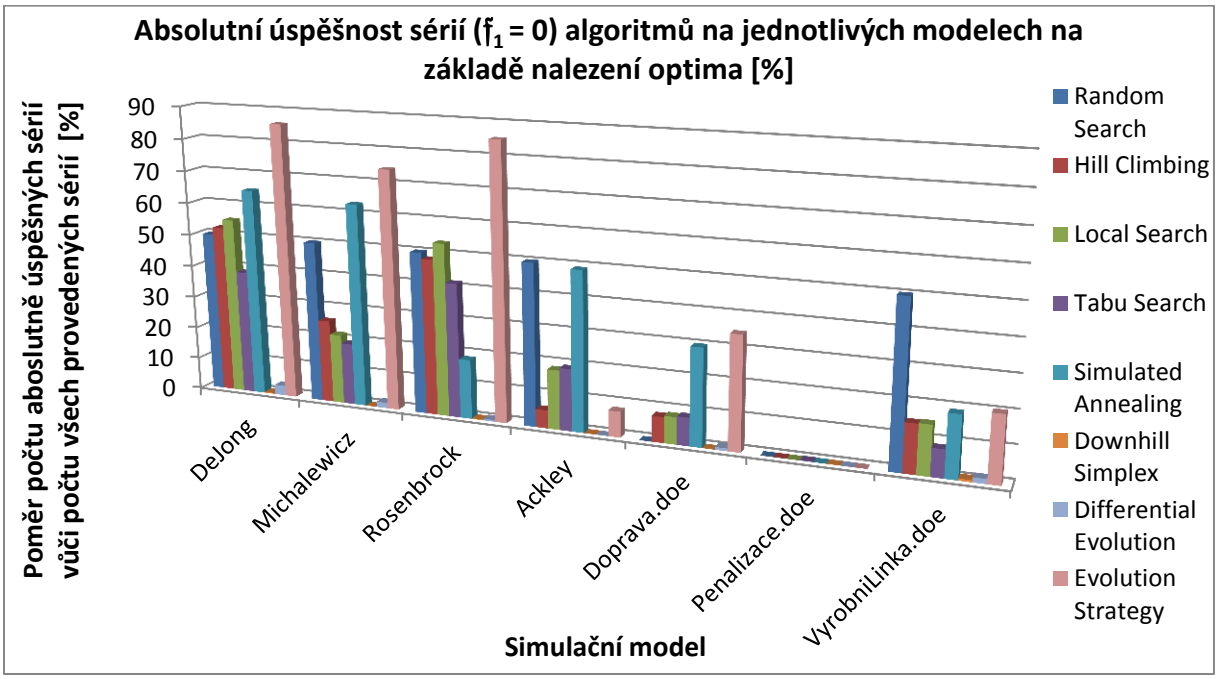
Při pohledu na následující podrobnější graf (viz Obr. 11-4) je patrné, že favorizovaný algoritmus Evolution Strategy měl problémy s multimodální „Ackleyho“ funkcí. Úspěšnost tohoto algoritmu byla velmi ovlivněna parametrem, který určoval počet jedinců v zápase. Nízké hodnoty tohoto parametru způsobily, že nenastalo vylepšování předchozí populace a nebyla vytvořena generace, která by konvergovala – putovala - do údolí globálního extrému. Jako druhý v pořadí úspěšnosti je literaturou doporučovaný univerzální algoritmus Simulated Annealing. Tento algoritmus díky své podstatě má možnost uniknout z lokálního extrému díky snižování a zvyšování pravděpodobnosti přijetí horšího řešení.

Může se zdát, že algoritmus Random Search je také úspěšným algoritmem. Ve skutečnosti je tomu ale tak, že ukazatel úspěšnosti je v našem případě ovlivněn počtem prováděných sérií. Série byly prováděny za účelem nalezení vhodného nastavení parametrů algoritmu. V případě algoritmu Random Search byly provedeny pouze 2 různá nastavení, protože algoritmus umožňuje nastavit pouze jeden parametr - generování stejných prvků - na booleovskou hodnotu Ano/Ne. Z toho plyne, že úspěšnost algoritmu může pro 2 různá nastavení – série - nabývat hodnoty 0 %, 50 % nebo 100 %.

Předcházející algoritmus výpočtu úspěšnosti by mohl být např. modifikován koeficientem upravujícím úspěšnost algoritmu v závislosti na počtu sérií nebo prováděním více sérií se stejným nastavením (provádění optimalizačních experimentů v sériích je ale časově velmi náročné, proto je tato práce spíše zaměřena na analýzu chování optimalizačního algoritmu u různých odlišných typů účelové funkce při různém nastavení jeho parametrů). Z podstaty algoritmu, kterým je čistě náhodné chování, je zřejmé, že algoritmus zvyšuje svoji úspěšnost s nárůstem počtu experimentů. Počet kroků, které algoritmus mohl provést, byl roven počtu všech možných řešení v prohledávaném prostoru. Tudíž jestliže byla nastavena možnost generovat pokaždé různé prvky a povrch účelové funkce nebyl příliš členitý, je zde velká pravděpodobnost, že algoritmus najde v malém prostoru globální optimum ve většině případů. S rostoucím prohledávaným prostorem ale bude jeho úspěšnost při hledání globálního optima značně snížena.

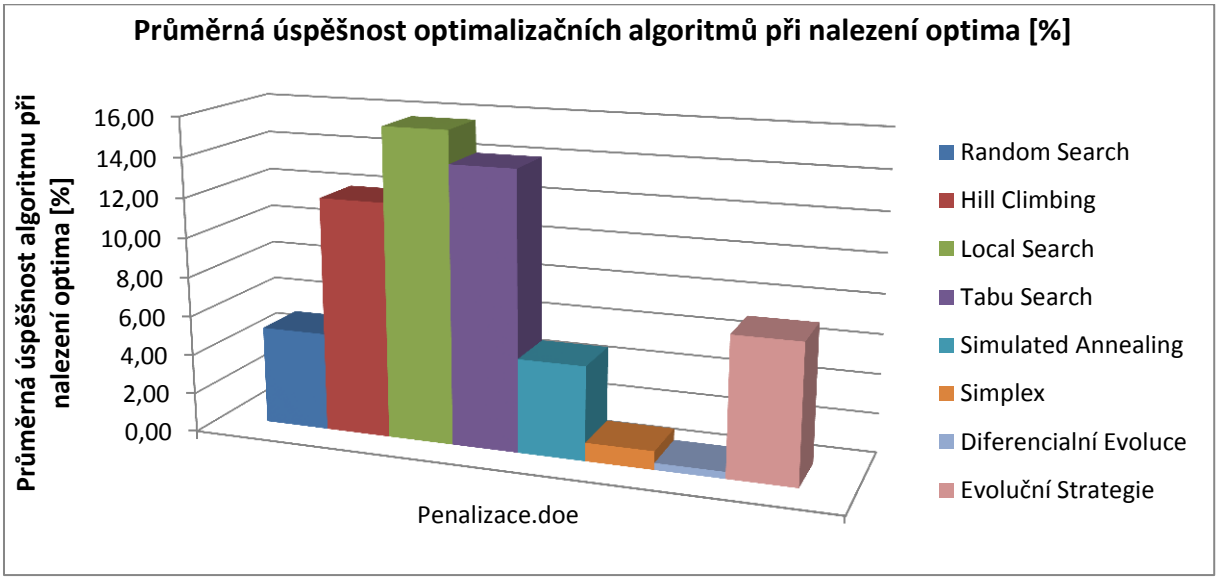
Chování optimalizačních algoritmů Hill Climbing, Local Search a Tabu Search je podobné díky základnímu pseudogradientnímu principu. Dobrou úspěšnost prokázaly algoritmy Simulated Annealing a zejména Evolution

Strategy. Random Search potvrzuje specifické vlastnosti při provedení dostatečného počtu optimalizačních experimentů. Toto fakt je potvrzen v následujícím sloupcovém grafu (viz Obr. 11-4).



Obr. 11-4 Absolutní úspěšnost sérií algoritmů na jednotlivých modelech na základě nalezení optima

Z grafu (viz Obr. 11-4) také vyplývá, že při velmi členitém povrchu účelové funkce – simulační model „Penalizace“ – mají všechny zvolené algoritmy při všech různých nastaveních problém nalézt globální optimum. Proto byla u tohoto simulačního modelu provedena analýza, která zjišťuje průměrnou úspěšnost všech sérií při nalezení globálního optima (viz předpis (11.1)). Výsledky této analýzy jsou uvedeny v následujícím grafu (viz Obr. 11-5).



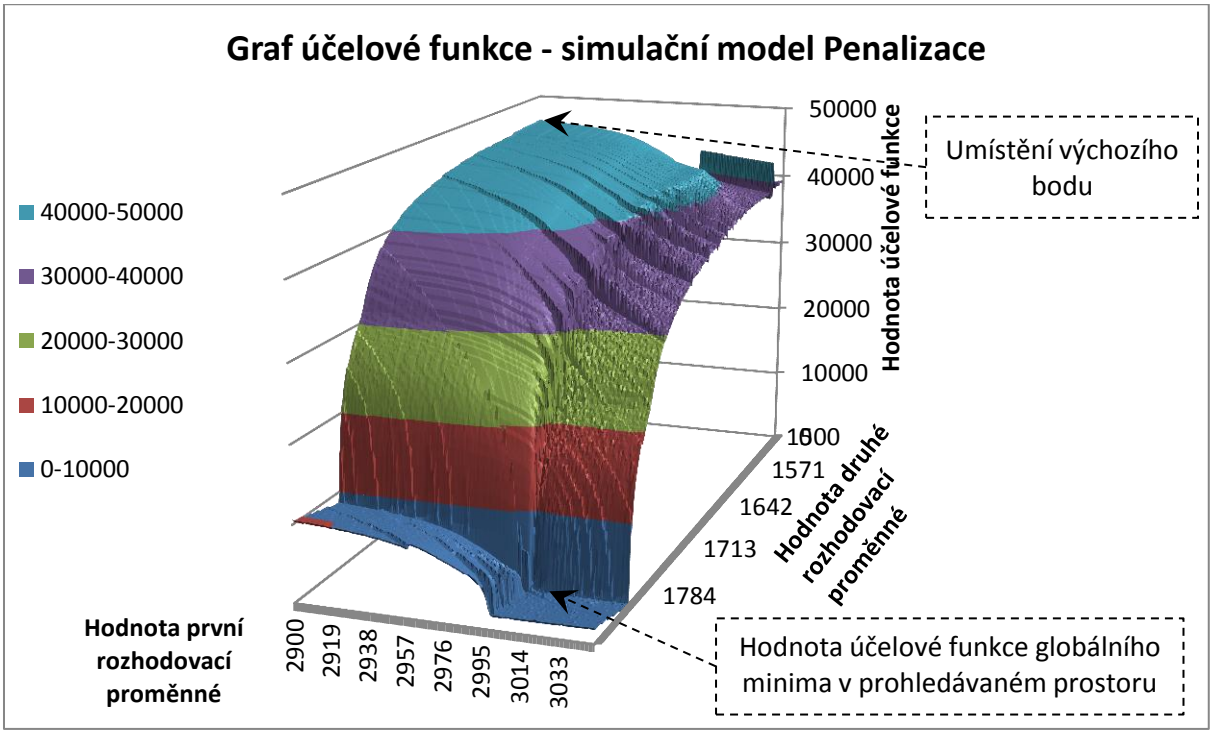
Obr. 11-5 Průměrná úspěšnost optimalizačních algoritmů na simulačním modelu „Penalizace“

Výchozí bodem (počátečním přípustným řešením) u simulačního modelu „Penalizace“ byl bod opačného globálního extrému (globální maximum), než který je hledán – globální minimum. Ze zmapovaného průběhu

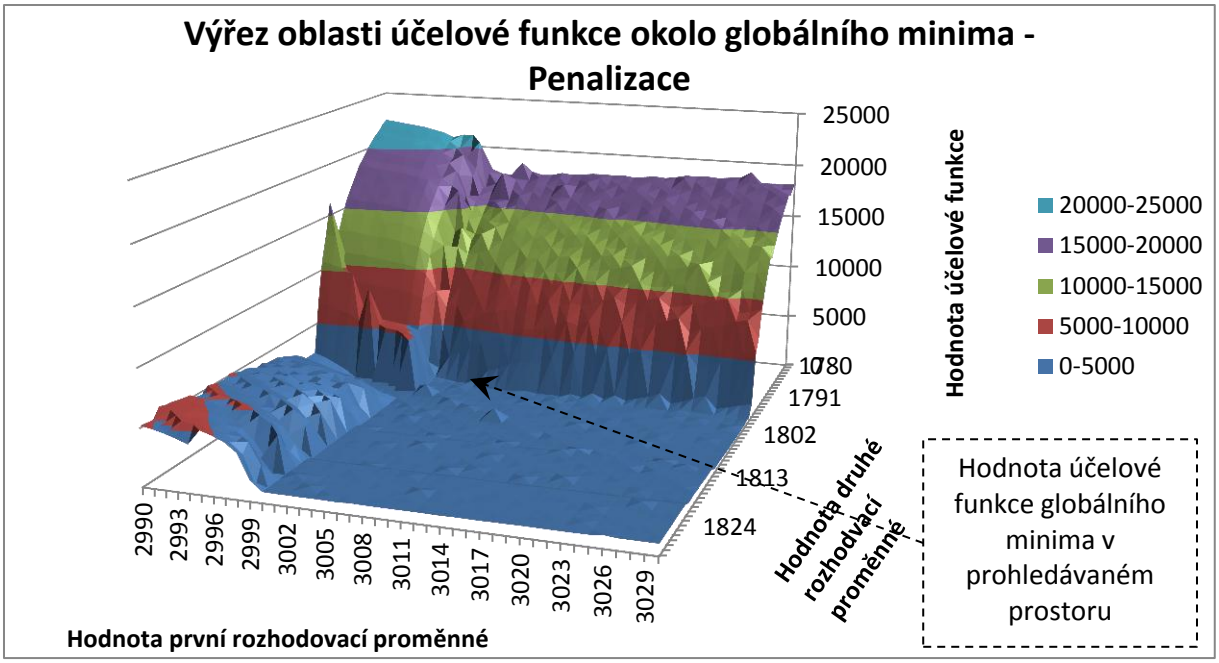
účelové funkce (viz Obr. 11-6) je vidět, že tento výchozí bod byl umístěn v rohu konkávního povrchu účelové funkce.

Z grafu (viz Obr. 11-5) je patrné, že nejlépe si vedly algoritmy založené na gradientním prohledávání, protože svažující se povrch účelové funkce okolo výchozí bodu velice prospívá tomuto typu gradientních algoritmů. Problém nastává u oblasti okolo globálního minima. Na výřezu účelové funkce okolo prostoru globálního minima (viz Obr. 11-7) jsou hodnoty poměrně stejné a algoritmus nemá dostupné informace o směru nejprudšího spádu, kterým by se chtěl vydat. Dále bylo definováno kritérium ukončení, že prohledávání optimalizačním algoritmem skončí tehdy, pokud bude nalezen globální extrém, který může nabývat hodnoty účelové funkce v tolerované odchylce jedné tisíciny od hodnoty účelové funkce globálního minima. Takový bod však poblíž globálního minima neexistuje a algoritmus se snažil prohledávat prostor dál. V praxi by taková tvrdá podmínka zřejmě definována nebyla a nalezená řešení v okolí globálního optima by byla uspokojivá. V tomto případě se ale výsledek nestal globálním optimem a úspěšnost podle první ohodnocující funkce byla neuspokojivá, což se odrazilo v grafu absolutní úspěšnosti algoritmů.

V grafu průměrné úspěšnosti optimalizačních algoritmů (viz Obr. 11-5) se znovu potvrdilo, že Evolution Strategy je poměrně úspěšným algoritmem ze všech zvolených algoritmů a je obecně využitelný pro různé typy účelových funkcí.

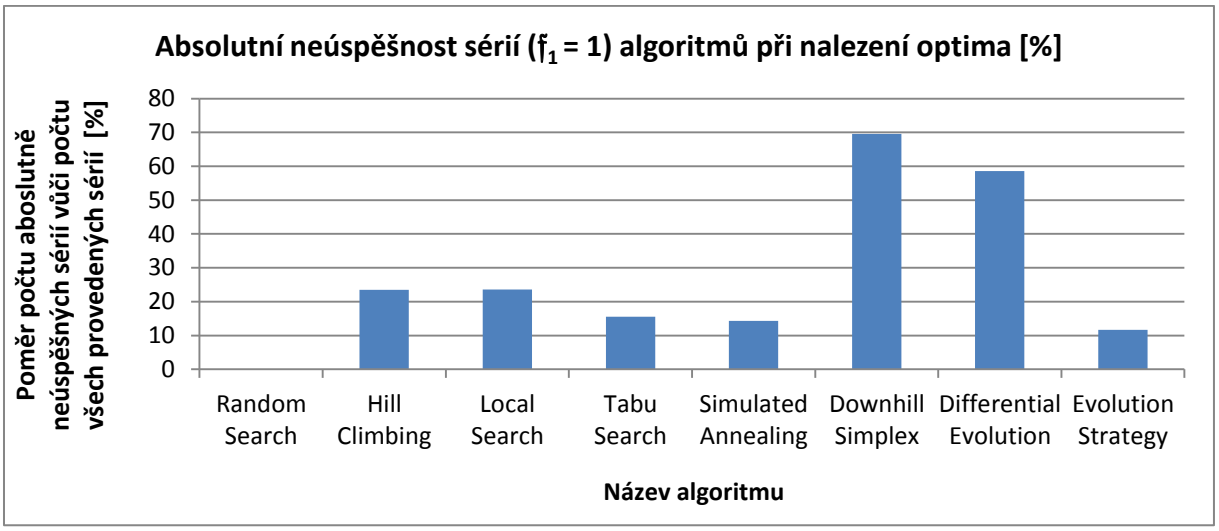


Obr. 11-6 Průběh účelové funkce modelu „Penalizace“ - umístění výchozího bodu, umístění globálního minima



Obr. 11-7 Průběh účelové funkce modelu „Penalizace“ - výřez oblasti účelové funkce okolo globálního minima

Protože předchozí grafy vyjadřovaly úspěšnosti jednotlivých algoritmů při nalezení globálního optima, měli bychom se v analýze chování optimalizačních algoritmů také zaměřit na **absolutní neúspěšnost** jednotlivých algoritmů na simulačních modelech. Absolutní neúspěšností se rozumí takový případ, kdy v sérii ze všech pokusů nebylo nalezeno ani jedno optimum ($f_1 = 1$). Následující graf vyjadřuje procentuální poměr počtu absolutně neúspěšných sérií vůči počtu ze všech provedených sérií (viz Obr. 11-8).



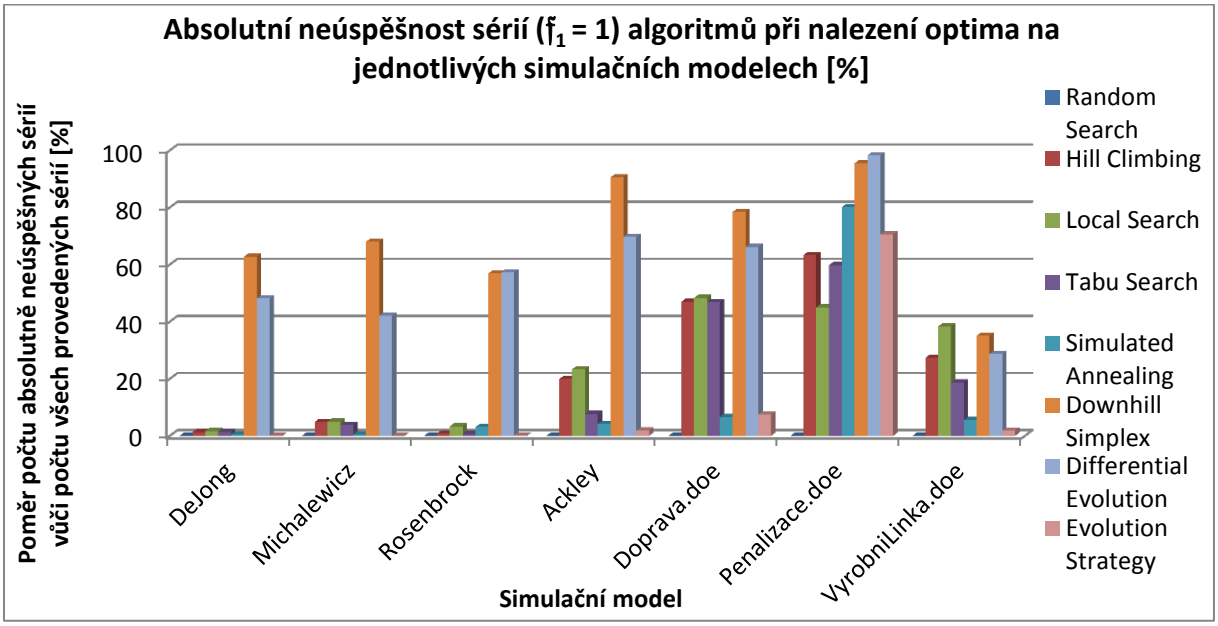
Obr. 11-8 Absolutní neúspěšnost algoritmů při nalezení optima

Algoritmus Random Search neprovedl ani jednu absolutně neúspěšnou sérii a proto také není ve sloupcovém grafu zobrazen příslušný sloupec, který by vyjadřoval hodnotu absolutní neúspěšnosti. Je to důsledek poměrně malého prohledávaného prostoru, povolení provedení velkého počtu simulačních experimentů a čistě náhodné podstaty tohoto algoritmu. Se zvyšujícím se rozměrem prohledávaného prostoru se však jeho neúspěšnost bude zvyšovat.

Není žádným překvapením, že nejhoršími algoritmy byly Downhill Simplex a Differential Evolution. Tyto algoritmy jsou přizpůsobeny zejména pro práci s reálnými čísly. Algoritmy založené na gradientu mají přibližně

stejnou neúspěšnost. Pro podrobnější analýzu chování algoritmů na jednotlivých simulačních modelech byl sestrojen následující graf (viz Obr. 11-9), který obsahuje detailnější informace o souhrnných hodnotách předchozího grafu (viz Obr. 11-8). Algoritmus Downhill Simplex se zachoval špatně z dříve uvedených důvodů dokonce i v případě jednoduchých průběhů účelových funkcí („De Jong“, „Rosenbrock“). To samé platí i pro algoritmus Differential Evolution, i když její výsledky při hledání globálního optima byly o něco lepší.

U simulačního modelu „Ackley“ je patrná menší absolutní neúspěšnost sérií algoritmu Tabu Search, oproti algoritmům Hill Climbing a Local Search.



Obr. 11-9 Absolutní neúspěšnost sérií algoritmů na jednotlivých modelech při nalezení optima

11.2 Kritérium - f_2 - Rozdíl lokálního extrému od optima

Kritérium f_2 je užitečné v případě, že se u žádného optimalizačního experimentu provedeného v sérii nenašlo globální optimum nebo prvek, který by svojí hodnotou účelové funkce byl v rozsahu definované tolerance od hodnoty účelové funkce globálního optima. V takovém případě by byla hodnota $f_1 = 1$ (viz kapitola 0). Pro tento případ je v následujícím algoritmu popsána podstata funkce, jež ohodnocuje rozdíl hodnoty účelové funkce nejlepšího nalezeného lokálního prvku v sérii vůči hodnotě účelové funkce globálního optima. Tato hodnota je také normalizována pomocí podílu, v jehož jmenovateli je absolutní hodnota rozdílu (vzdálenost) hodnoty účelové funkce globálního optima a hodnoty účelové funkce nejhoršího prvku v prohledávaném prostoru (v případě minimalizace účelové funkce je to globální maximum a v případě maximalizace účelové funkce je to globální minimum). Opět platí $f_2 \in [0,1]$. Snahou je **minimalizovat** hodnotu f_2. Navrácená hodnota je $f_2 = 0$ v případě, kdy hodnota účelové funkce nejlepšího nalezeného prvku (nebo nejlepšího prvku z množiny optim) je rovna hodnotě účelové funkce globálního optima. V takovém případě bude ale rozhodující kritérium f_1 (viz kapitola 11.1), a zajímavá bude dále uvedená hodnota f_3 (míra rozptylu výsledků).

$f_2 \leftarrow \text{AssignFitnessBest}(X^*, \mathbf{X}^*)$		
Funkce, jejímž výstupem je skalární normovaná hodnota v rozsahu $f_2 \in [0,1]$. Tato hodnota představuje ohodnocení rozdílu hodnoty účelové funkce nejlepšího nalezeného lokálního extrému (prvek je také mimo toleranci od hodnoty účelové funkce) od hodnoty účelové funkce globálního optima v konkrétní sérii vzhledem k rozdílu hodnoty účelové funkce nejlepšího (globální optimum) a nejhorší prvku uvnitř prohledávaného prostoru		
Vstup:	X^*:	Seznam nalezených optim v konkrétní sérii
Vstup:	$\mathbf{X}^*$:	Globální optimum $\mathbf{X}^*$ v prohledávaném prostoru

Data:	$F(\mathbf{X})$: Účelová funkce
Data:	$\mathbf{X}_{\mathrm{Worst}}$: Nejhorší prvek v prohledávaném prostoru
Výstup:	$\mathfrak{f}_2$: Skalární normovaná hodnota ohodnocení kvality nejlepšího nalezeného řešení v množině optim, které je různé od optima a tolerance vůči optimu

1 **begin**

2 $X^* \longleftarrow \mathrm{Sort}_{\mathrm{a}}\left(X^*, \mathrm{CF}_{F(\mathbf{X})}\right)$;
(*vzestupné setřídění seznamu optim podle účelové funkce pro případ minimalizace účelové funkce – nultý prvek je nejlepší; maximalizace účelové funkce – sestupné třídění seznamu*)

3 **result** $\longleftarrow \frac{|F(\mathbf{X}^*)-F(X^*[0])|}{|F(\mathbf{X}^*)-F(\mathbf{X}_{\mathrm{Worst}})|}$; //normování

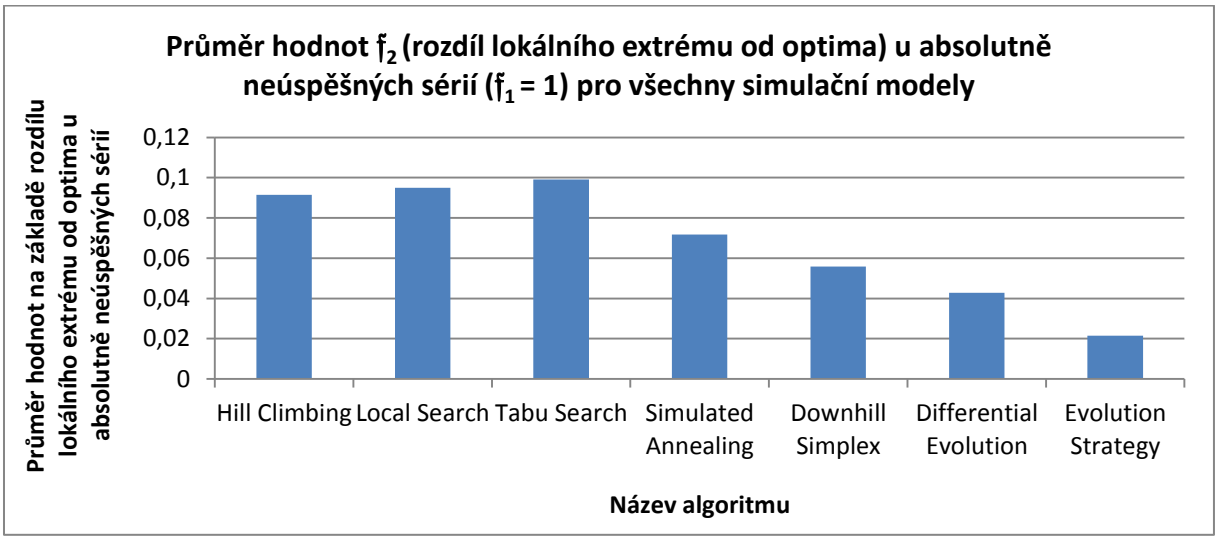
4 **end**;

Algoritmus 11-3 AssignFitnessBest

Ohodnocení výsledků sérií podle $\mathfrak{f}_2$ - Rozdíl lokálního extrému od optima

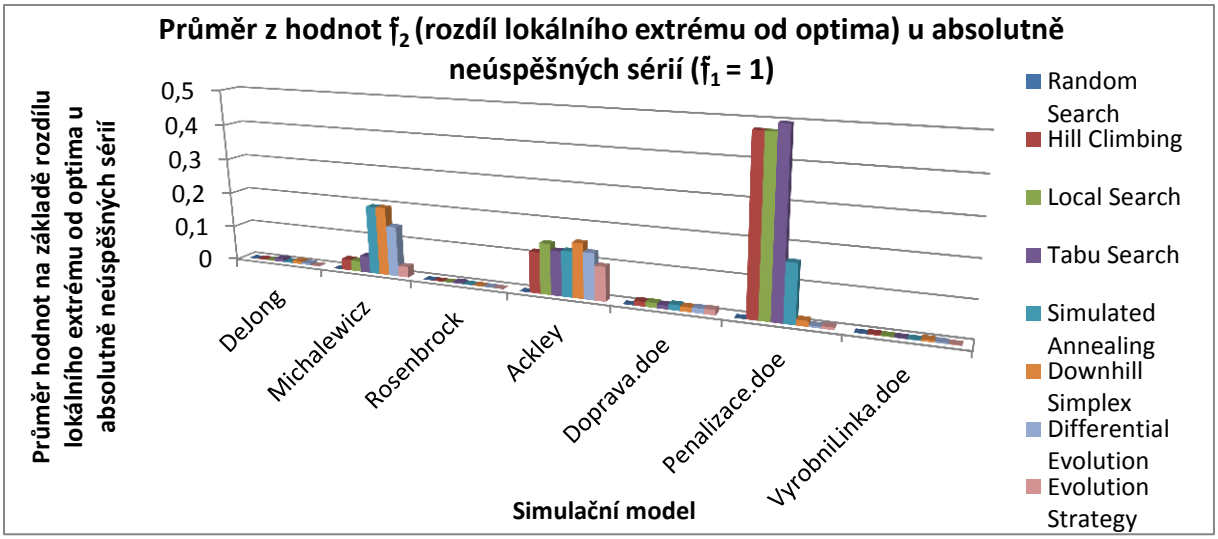
V následujícím grafu (viz Obr. 11-10) jsou zobrazeny průměry hodnot na základě $\mathfrak{f}_2$, tj. ohodnocení rozdílu lokálního extrému od globálního optima u optimalizačního algoritmu pro všech sedm simulačních modelů. Ve výběru jsou zahrnuty pouze výsledky takových sérií, které byly naprosto neúspěšné při hledání optima, tj., $\mathfrak{f}_1 = 1$. V grafu se nevyskytuje žádný sloupec u algoritmu Random Search z důvodu, že algoritmus neprovedl ani jednu absolutně neúspěšnou sérii (algoritmus měl k dispozici dostatek provedení simulačních experimentů u menších prohledávaných prostorů). Z analýzy uvedené v grafu je patrné, že v případě absolutně neúspěšných sérií, algoritmy Hill Climbing, Local Search a Tabu Search poskytovaly horší výsledky při hledání optima. Znamená to, že algoritmy poskytovaly takové prvky, které svojí hodnotou účelové funkce nebyly blízké hodnotě účelové funkce globálního optima. Nejlépe se podle kritéria $\mathfrak{f}_2$ projevuje algoritmus Evoluční strategie.

Poznámka: Pokud se v grafu nenachází některý simulační model, optimalizační algoritmus nebo nějaká hodnota parametru, znamená to, že některá z předchozích vyjmenovaných položek nesplnila kritéria výběru (např. absolutní neúspěšnost) a nebyla zahrnuta do hodnocení.



Obr. 11-10 Průměr hodnot $\mathfrak{f}_2$ (rozdíl lokálního extrému od optima) u absolutně neúspěšných sérií pro všechny simulační modely

Výsledky v grafu (viz Obr. 11-11) odpovídají předchozímu hodnocení podle $\mathfrak{f}_1$.



Obr. 11-11 Průměr z hodnot $\tilde{f}_2$ (rozdíl lokálního extrému od optima) u absolutně neúspěšných sérií

11.3 Kritérium - $\tilde{f}_3$ - Vzdálenost kvartilů

V pořadí již třetí ohodnocující funkce navrací normalizovanou hodnotu ($\tilde{f}_3 \in [0,1]$), která vyjadřuje, jaké vzdálenosti jsou mezi jednotlivými kvartily v sérii. Souhrnně řečeno nám tato funkce poskytuje představu o tom, jaký byl rozsah hodnot účelové funkce u výsledků v konkrétní sérii. Pro účely hodnocení jsou využity váhy, které penalizují výsledky umístěné v příslušném kvartilu. Váhové koeficienty pro jednotlivé rozsahy mezi hranicemi jednotlivých kvartilů v případě **minimalizace** účelové funkce jsou zaznamenány v následující tabulce (viz Tabulka 11-2). Suma vah je rovna jedné. Následující funkce (viz (11.2)) vyjadřuje ohodnocení kvality nalezených výsledků v konkrétní sérii pro případ minimalizace účelové funkce:

$$\tilde{f}_3 = \frac{|Q_1 - F(\mathbf{X}^*)| + w_{4\tilde{f}_3}|Q_1 - Q_2| + w_{3\tilde{f}_3}|Q_2 - Q_3| + w_{2\tilde{f}_3}|Q_3 - Q_4| + w_{1\tilde{f}_3}|Q_4 - Q_5|}{|F(\mathbf{X}^*) - F(\mathbf{X}_{\mathrm{Worst}})|} \tag{11.2}$$

kde:

- $w_{4\tilde{f}_3}$ … Hodnota váhy (penalizace) pro interval mezi minimem (Q_1) a dolním kvartilem (Q_2) hodnot účelové funkce výsledků poskytnutých optimalizačním algoritmem pro jednotlivou sérii.
- $w_{3\tilde{f}_3}$ … Hodnota váhy pro interval mezi dolním kvartilem (Q_2) a mediánem (Q_3) hodnot účelové funkce výsledků poskytnutých optimalizačním algoritmem pro jednotlivou sérii.
- $w_{2\tilde{f}_3}$ … Hodnota váhy pro interval mezi mediánem (Q_3) a horním kvartilem (Q_4) hodnot účelové funkce výsledků poskytnutých optimalizačním algoritmem pro jednotlivou sérii.
- $w_{1\tilde{f}_3}$ … Hodnota váhy pro interval mezi horním kvartilem (Q_4) a maximem (Q_5) hodnot účelové funkce výsledků poskytnutých optimalizačním algoritmem pro jednotlivou sérii.
- $F(\mathbf{X}_{\mathrm{Worst}})$ … Hodnota účelové funkce nejhoršího prvku v prohledávaném prostoru.
- $F(\mathbf{X}^*)$ … Hodnota účelové funkce nejlepšího prvku – globálního optima v prohledávaném prostoru.

Snahou je, aby charakteristiky krabicového grafu byly co nejblíže globálního optima $\mathbf{X}^*$ (v případě minimalizace účelové funkce - globálního minima).

Rozsah		Označení váhy	Hodnota váhy
Dolní hranice	**Horní hranice**		
Minimum	Dolní kvartil	$w_{1\mathfrak{f}_3}$	0,05
Dolní kvartil	Medián	$w_{2\mathfrak{f}_3}$	0,1
Medián	Horní kvartil	$w_{3\mathfrak{f}_3}$	0,25
Horní kvartil	Maximum	$w_{4\mathfrak{f}_3}$	0,6
		Suma vah	1

Tabulka 11-2 Váhy pro jednotlivé rozsahy u hodnot účelové funkce poskytnutých výsledků jedné série v případě minimalizace účelové funkce

Protože charakteristiky krabicového grafu jsou fixní, to znamená, že při **maximalizaci** účelové funkce je snahou, aby jednotlivé hodnoty kvartilů byly co nejblíže globálního maxima, pro výpočet ohodnocení následující funkce je proto použita předchozí transformovaná funkce (viz (11.3)).

$$\mathfrak{f}_3 = \frac{|F(\mathbf{X}^*) - Q_5| + w_{1\mathfrak{f}_3}|Q_1 - Q_2| + w_{2\mathfrak{f}_3}|Q_2 - Q_3| + w_{3\mathfrak{f}_3}|Q_3 - Q_4| + w_{4\mathfrak{f}_3}|Q_4 - Q_5|}{|F(\mathbf{X}^*) - F(\mathbf{X}_{\mathrm{Worst}})|} \tag{11.3}$$

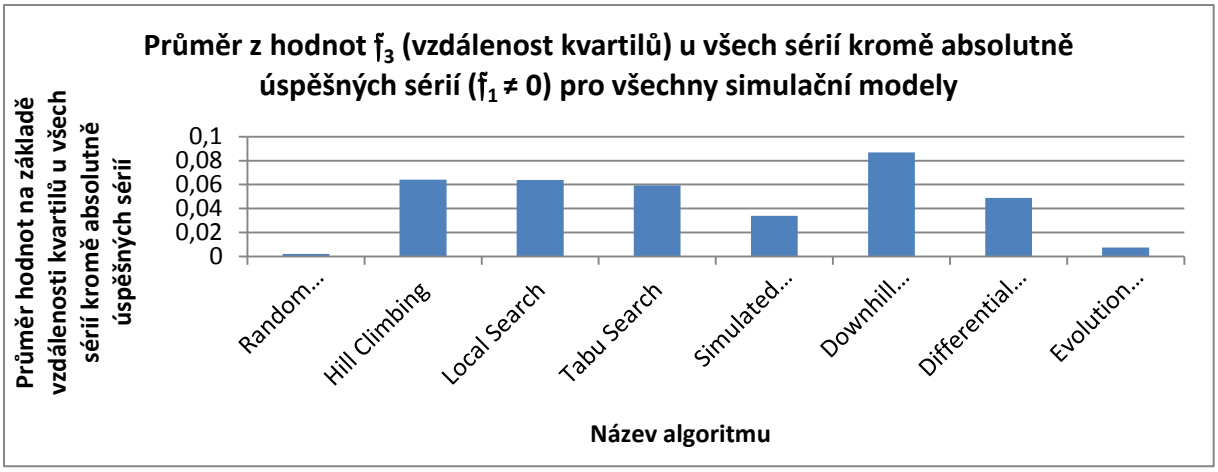
V praxi ve většině případů nebude možné globální optimum spolehlivě určit. V tomto případě by namísto hodnoty účelové funkce globálního optima mohla být použita hodnota účelové funkce nejlepšího nalezeného prvku (řešení) ze všech simulačních experimentů.

Ohodnocení výsledků sérií podle $\mathfrak{f}_3$ - vzdálenost kvartilů

V následujícím grafu (viz Obr. 11-12) jsou zobrazeny průměrné hodnoty kritéria $\mathfrak{f}_3$ (vzdálenost kvartilů, tj., měřítko rozptylu výsledků) u poskytovaných výsledků optimalizačních experimentů jednotlivých optimalizačních algoritmů pro všech sedm simulačních modelů. Tento ukazatel vyjadřuje míru kvality výsledků poskytnutých optimalizačním algoritmem v sérii. Do výběru jsou zahrnuty všechny provedené série kromě absolutně úspěšných sérií, tj. $\mathfrak{f}_1 \neq 0$.

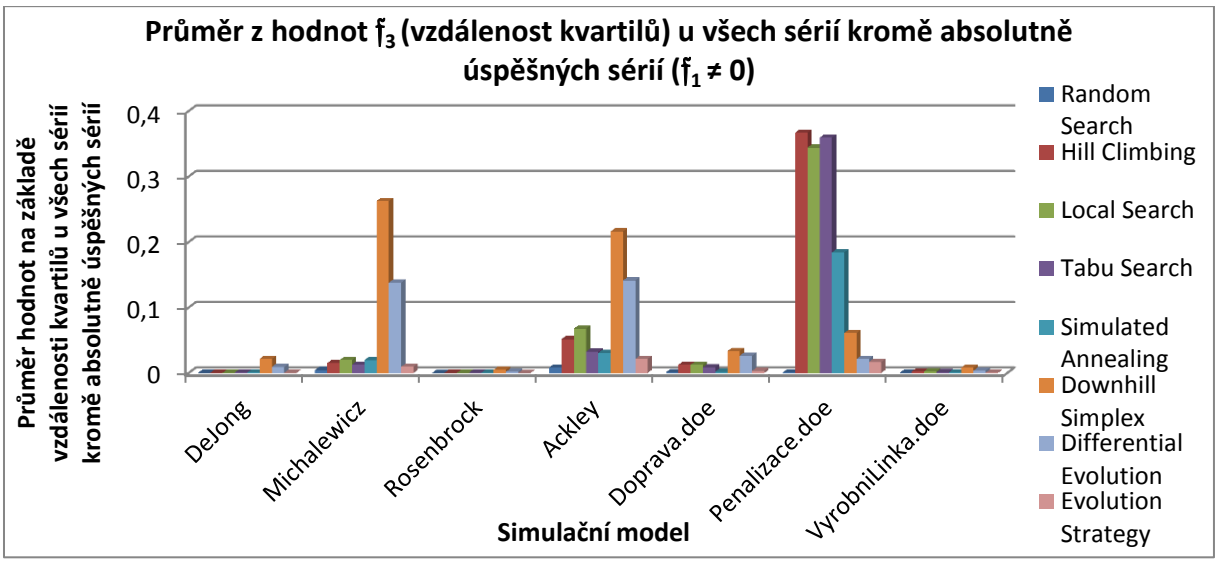
V případě absolutně úspěšné série, by hodnota ukazatele nabývala nuly, tj., $\mathfrak{f}_1 = 0 \rightarrow \mathfrak{f}_3 = 0$ (u každého optimalizačního experimentu bylo nalezeno globální optimum). Jako i v případě minulých kritérií, je snahou minimalizovat hodnoty v grafu, protože větší hodnoty $\mathfrak{f}_3$ představují spíše neúspěšnost optimalizačního algoritmu. Toto kritérium v sobě zahrnuje penalizaci odlehlejších bodů od globálního optima pomocí stanovených, v tomto případě subjektivních, vah. Hodnota těchto vah závisí na preferenci kvality poskytovaného řešení.

Algoritmus Downhill Simplex poskytoval v průměru nejhorší výsledky (rozsah hodnot účelové funkce výsledků poskytnutých optimalizačním algoritmem) ze všech testovaných optimalizačních algoritmů díky svému zaokrouhlování souřadnic a tím způsobeného odchýlení od vypočteného směru. Pseudogradientní přístupy nalezly zhruba stejně kvalitní řešení. V případě algoritmu Simulated Annealing je jeho kvalita podle $\mathfrak{f}_3$ vzhledem k algoritmu Evolution Strategy několikrát horší.



Obr. 11-12 Průměr z hodnot f_3 (vzdálenost kvartilů) u všech sérií kromě absolutně úspěšných sérií pro všechny simulační modely

Následující graf (viz Obr. 11-13) poskytuje detailnější pohled na kritérium f_3 u jednotlivých simulačních modelů.



Obr. 11-13 Průměr z hodnot f_3 (vzdálenost kvartilů) u všech sérií kromě absolutně úspěšných sérií

11.4 Kritérium - f_4 - Rychlost nalezení optima

Kritérium f_4 hodnotí rychlost nalezení optima, tj. počet vyhodnocení účelové funkce (počet provedených simulačních experimentů), které byly provedeny do doby, než bylo nalezeno optimum, popřípadě nejlepší prvek (řešení) ze všech simulačních experimentů provedených v rámci optimalizačního experimentu.

Kvartilové charakteristiky krabicového grafu jsou spočteny z hodnot počtu provedených simulačních experimentů do nalezení nejlepšího prvku ze všech optimalizačních experimentů série. Na základě těchto charakteristik krabicového grafu je spočtena hodnota f_4.

Následující funkce f_4 (viz (11.4) vyjadřuje ohodnocení rychlosti nalezení optim (globálních i lokálních) v konkrétní sérii. Hodnota funkce je normalizována ($f_4 \in [0,1]$) a je snahou vždy **minimalizovat** hodnotu f_4. Váhy pro rozsahy mezi jednotlivými kvartily jsou stejné jako v předchozím případě třetího kritéria.

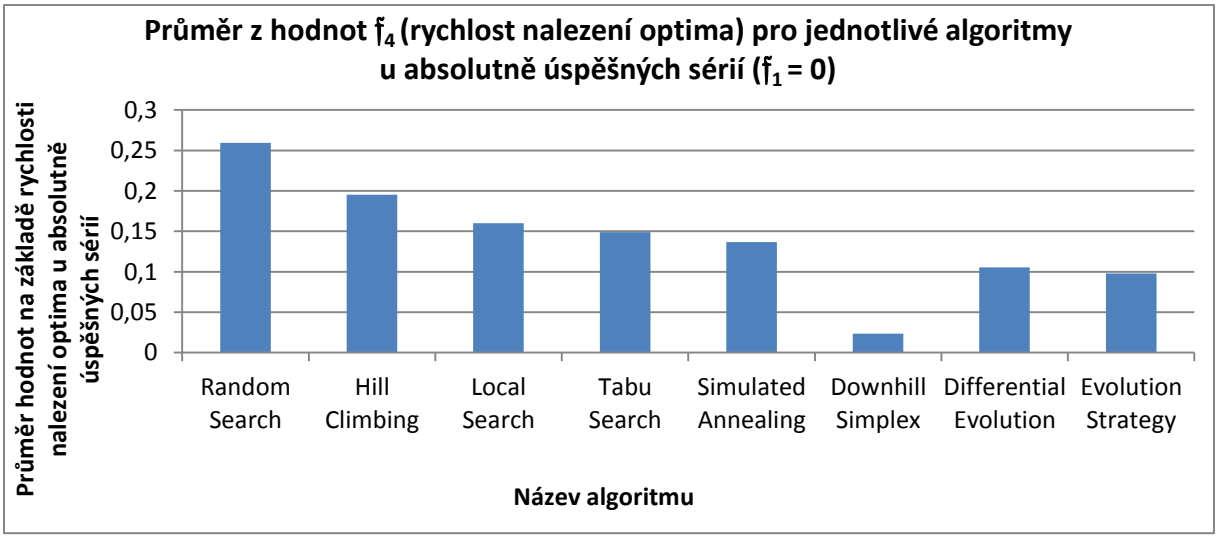
$$f_4 = \frac{|Q_1 - 1| + w_{4f_4}|Q_1 - Q_2| + w_{3f_4}|Q_2 - Q_3| + w_{2f_4}|Q_3 - Q_4| + w_{1f_4}|Q_4 - Q_5|}{m_{\tilde{x}}} \quad (11.4)$$

kde:

- $w_{4\mathrm{f}_4}$ … Hodnota váhy pro interval mezi minimem (Q_1) a dolním kvartilem (Q_2) hodnot, které vyjadřují počet provedených simulačních experimentů do nalezení optima u všech optimalizačních experimentů v konkrétní sérii.
- $w_{3\mathrm{f}_4}$ … Hodnota váhy pro interval mezi dolním kvartilem (Q_2) a mediánem (Q_3) hodnot, které vyjadřují počet provedených simulačních experimentů do nalezení optima u všech optimalizačních experimentů v konkrétní sérii.
- $w_{2\mathrm{f}_4}$ … Hodnota váhy pro interval mezi mediánem (Q_3) a horním kvartilem (Q_4) hodnot, které vyjadřují počet provedených simulačních experimentů do nalezení optima u všech optimalizačních experimentů v konkrétní sérii.
- $w_{1\mathrm{f}_4}$ … Hodnota váhy pro interval mezi horním kvartilem (Q_4) a maximem (Q_5) hodnot, které vyjadřují počet provedených simulačních experimentů do nalezení optima u všech optimalizačních experimentů v konkrétní sérii.
- $m_{\tilde{X}}$ ... Počet prvků (možných – přípustných - řešení) v prohledávaném prostoru $\tilde{X}$.

Ohodnocení výsledků sérií podle f_4 – rychlost nalezení optima

V následujícím grafu (viz Obr. 11-14) jsou zobrazeny průměrné hodnoty f_4 (rychlost nalezení optima nebo nejlepšího výsledku optimalizačního experimentu) u jednotlivých optimalizačních algoritmů. Do výběru jsou zahrnuty všechny provedené absolutně série optimalizačních algoritmů. V kategorii nejrychlejších algoritmů se nejlépe umístil Downhill Simplex. Tento algoritmus dosahoval ale špatných výsledků při hledání globálního optima. U tohoto optimalizačního algoritmu je zapotřebí provést hlubší analýzu jeho chování a provést případné modifikace. Podle průměru z hodnot f_4, se jako další (rychlé) umístily algoritmy Differential Evolution a Evolution Strategy.

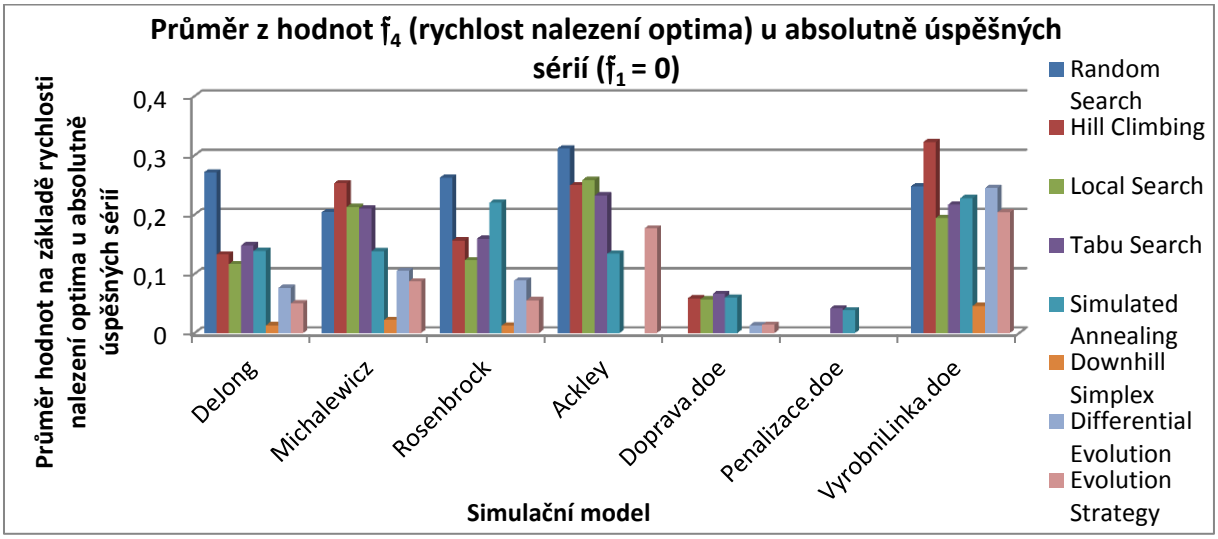


Obr. 11-14 Průměr z hodnot f_4 (rychlost nalezení optima) pro jednotlivé algoritmy

Při detailnějším pohledu na jednotlivé simulační modely vzhledem k optimalizačním algoritmům (viz Obr. 11-15), lze říci, že algoritmy založené na gradientním prohledávání dosahují v rychlosti nalezení optima téměř stejných výsledků ohodnocení. Vzhledem k celkovému hodnocení úspěšnosti při hledání optima a rychlosti nalezení optima se jeví jako velmi efektivní algoritmus Evolution Strategy. Jako druhý podle hodnocení úspěšnosti při nalezení optima se jeví algoritmus Simulated Annealing. Tento algoritmus ale potřebuje provést více simulačních experimentů k nalezení bodu globálního optima. Je to dáno jeho podstatou, která je odvozena od podstaty pseudogradientních metod.

U simulačního modelu „Penalizace“ dosáhly pouze algoritmy Tabu Search a Simulated Annealing absolutně úspěšně série. U simulačního modelu „Doprava“ algoritmus Random Search ani Downhill Simplex nebyly schopné dosáhnout ani jedné absolutně úspěšné série.

Algoritmus Random Search je založen na čistě náhodném prohledávání. Pokud bude prohledávaný prostor rozlehlý a u kritéria ukončení by bylo povoleno malé množství simulačních experimentů, bude v takových případech velká pravděpodobnost, že algoritmus Random Search bude při hledání globálního optima značně neefektivní.



Obr. 11-15 Průměr z hodnot $\tilde{f}_4$ (rychlost nalezení optima) u absolutně úspěšných sérií

11.5 Kritérium - $\tilde{f}_5$ – Konvergence

Kritérium $\tilde{f}_5$ (viz (11.5)) vyjadřuje, jak simulační experimenty u všech optimalizačních experimentů v konkrétní sérii konvergovaly ke globálnímu optimu, respektive míru, jak daleko jsou hodnoty účelové funkce vygenerovaných prvků (řešení, simulačních experimentů) od hodnoty účelové funkce globálního optima. Hodnota funkce je normalizovaná ($\tilde{f}_5 \in [0,1]$). **Minimální** hodnoty této funkce vedou k závěru, že funkce rychle konverguje, což během optimalizačního procesu je jistě žádané. Na druhé straně minimální hodnota vypovídá o **elitářství** nejlepších prvků, které mohou vést k předčasné konvergenci algoritmu. Hodnoty vah pro rozsahy mezi jednotlivými kvartily jsou stejné jako v předchozím případě. Následující funkce (viz (11.5)) je užita pro ohodnocení konvergence série pro případ minimalizace účelové funkce:

$$\tilde{f}_5 = \frac{|Q_1 - F(\mathbf{X}^*)| + w_{4\tilde{f}_5}|Q_1 - Q_2| + w_{3\tilde{f}_5}|Q_2 - Q_3| + w_{2\tilde{f}_5}|Q_3 - Q_4| + w_{1\tilde{f}_5}|Q_4 - Q_5|}{|F(\mathbf{X}^*) - F(\mathbf{X}_{\text{Worst}})|} \qquad (11.5)$$

kde:

- $w_{4\tilde{f}_5}$ … Hodnota váhy pro interval mezi minimem (Q_1) a dolním kvartilem (Q_2) hodnot účelové funkce generovaných řešení optimalizačním algoritmem pro jednotlivou sérii.
- $w_{3\tilde{f}_5}$ … Hodnota váhy pro interval mezi dolním kvartilem (Q_2) a mediánem (Q_3) hodnot účelové funkce generovaných řešení optimalizačním algoritmem pro jednotlivou sérii.
- $w_{2\tilde{f}_5}$ … Hodnota váhy pro interval mezi mediánem (Q_3) a horním kvartilem (Q_4) hodnot účelové funkce generovaných řešení optimalizačním algoritmem pro jednotlivou sérii.
- $w_{1\tilde{f}_5}$ … Hodnota váhy pro interval mezi horním kvartilem (Q_4) a maximem (Q_5) hodnot účelové funkce generovaných řešení optimalizačním algoritmem pro jednotlivou sérii.
- $F(\mathbf{X}_{\text{Worst}})$ … Hodnota účelové funkce nejhoršího prvku v prohledávaném prostoru.
- $F(\mathbf{X}^*)$ … Hodnota účelové funkce nejlepšího prvku – globálního optima v prohledávaném prostoru.

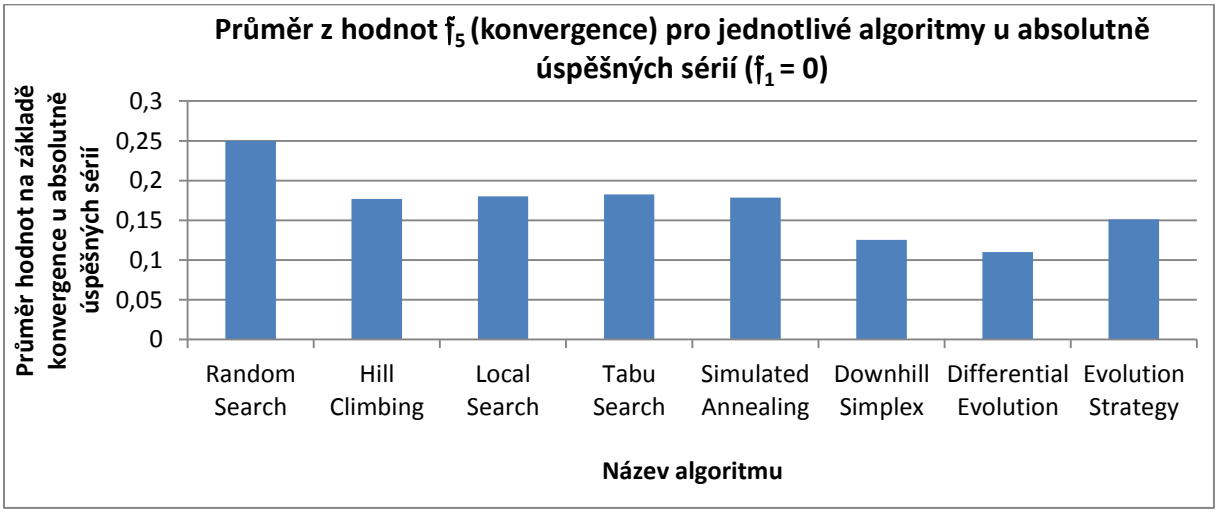
Protože charakteristiky krabicového grafu jsou fixní, to znamená, že při maximalizaci účelové funkce je snahou, aby jednotlivé hodnoty kvartilů byly co nejblíže globálního maxima, pro výpočet ohodnocení následující funkce je proto použita předchozí transformovaná funkce (viz (11.6)).

$$\mathrm{f}_5 = \frac{|F(\mathbf{X}^*) - Q_5| + w_{1\mathrm{f}_5}|Q_1 - Q_2| + w_{2\mathrm{f}_5}|Q_2 - Q_3| + w_{3\mathrm{f}_5}|Q_3 - Q_4| + w_{4\mathrm{f}_5}|Q_4 - Q_5|}{|F(\mathbf{X}^*) - F(\mathbf{X}_{\mathrm{Worst}})|} \tag{11.6}$$

Ohodnocení výsledků sérií podle f_5 – konvergence

Následující graf (viz Obr. 11-16) zachycuje průměrné hodnoty pátého kritéria (konvergence) u jednotlivých optimalizačních algoritmů ze všech absolutně úspěšných sérií. Snahou je minimalizovat hodnoty v grafu. Na druhou stranu velikost tohoto ukazatele určuje diverzitu generovaných potenciálních řešení v jednotlivých simulačních experimentech. Není tedy překvapením, že největší ukazatel má algoritmus Random Search. Tento algoritmus je založen na principu metod Monte Carlo a nedá se mluvit o strategické konvergenci ke globálnímu optimu. Z provedených sérií je také vidět shodnost podstaty algoritmů založených na pseudogradientním přístupu. Algoritmy Hill Climbing, Local Search, Tabu Search a Simulated Annealing dosahují přibližně stejné hodnoty kritéria konvergence.

Z hodnot sloupců u algoritmů Downhill Simplex a Differential Evolution lze usuzovat, že jejich konvergence je lepší než u ostatních optimalizačních algoritmů. Musíme mít ale na paměti, že do výběru byly zahrnuty pouze absolutně úspěšné série. Pokud tedy srovnáme grafy procentuální podíly počtu absolutně úspěšných (viz Obr. 11-3) a absolutně neúspěšných sérií (viz Obr. 11-8) lze konstatovat, že tyto algoritmy dosahovaly při hledání optima velice podprůměrných úspěchů. V případě **absolutní** úspěšnosti byly hodnoty u Downhill Simplexu 0.17 % a Differential Evolution 1.1 % **oproti** 44 % úspěšnosti algoritmu Evolution Strategy. Je také možné, že tento neúspěch mohl být zapříčiněn špatně nastavenými parametry algoritmu (jejich vliv byl ale značně omezen výběrem pouze absolutně úspěšných sérií). Z tohoto důvodu byla analyzována jednotlivá nastavení optimalizačních algoritmů v závislosti na výsledcích optimalizačních experimentů.

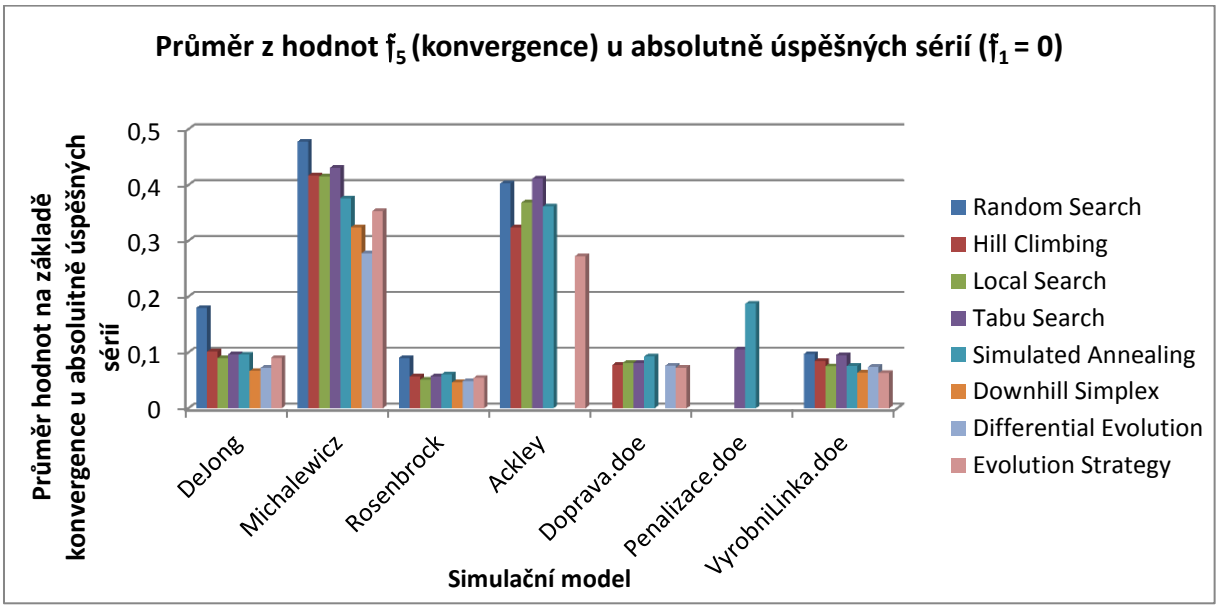


Obr. 11-16 Průměr z hodnot f_5 (konvergence) pro jednotlivé algoritmy u absolutně úspěšných sérií

V dalším grafu (viz Obr. 11-17) je zobrazen detailnější pohled na jednotlivé optimalizační algoritmy vzhledem k simulačním modelům.

U simulačního modelu „Penalizace“ pouze dva algoritmy – Tabu Search a Simulated Annealing dosáhly absolutně úspěšných sérií při hledání globálního optima.

V případě Tabu Search byla konvergence lepší než u algoritmu Simulated Annealing. Není překvapením, že algoritmus Random Search dosahoval nejhorší konvergence ze všech vybraných algoritmů. Největší problémy s konvergencí měly pseudogradientní algoritmy u simulačního modelu „Ackley“ a „Michalewicz“.



Obr. 11-17 Průměr z hodnot $\bar{f}_5$ (konvergence) u absolutně úspěšných sérií

12 Metodika výběru vhodného nastavení parametrů optimalizačního algoritmu

Při výběru vhodného nastavení parametrů optimalizačních algoritmů byly specifikovány tří různé pohledy na provedené série na vybraných simulačních modelech:

- Vhodné nastavení konkrétního parametru zvoleného algoritmu pro zvolený simulační modely – vliv výše hodnoty konkrétního parametru na chování optimalizačního algoritmu při hledání globálního optima.
- Vhodné nastavení všech parametrů zvoleného algoritmu pro zvolený simulační modely – nalezení nejlepšího nastavení parametrů (série) u konkrétních simulačních modelů v závislosti na stanovených preferencích jednotlivých kritérií ze všech sérií.
- Vhodné „univerzální“ nastavení všech parametrů zvoleného algoritmu pro všechny vybrané simulační modely – nalezení univerzálního nastavení pro všechny zvolené simulační modely v závislosti na stanovených preferencích jednotlivých kritérií ze všech sérií.

12.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Prvním krokem při hledání vhodného nastavení konkrétního parametru optimalizačního algoritmu bylo provést ohodnocení všech testovaných sérií na základě předchozích specifikovaných ohodnocujících funkcí. Dále byl proveden výběr pouze těch sérií u jednotlivého algoritmu, jejichž hodnota prvního kritéria, tj. „Počet nalezených optim nebo hodnot blízkých optimu“ je rovna nejlepší nalezené hodnotě tohoto kritéria ze všech testovaných nastavení.

U všech optimalizačních experimentů bylo zjištěno, že nejnižší nalezená hodnota prvního kritéria f_1 byla u všech algoritmů rovna tj. $f_1 = 0$. Znamená to tedy, že u všech provedených sérií pro každý algoritmus se vyskytla minimálně jedna sérii, která měla 100 % úspěšnost při hledání globálního optima.

Podmínka výběru nejvyšší nalezené úspěšnosti u vybraného algoritmu byla vybrána záměrně z toho důvodu, že některé algoritmy ani v nejlepších sériích nemusely být schopny v prohledávaném prostoru nalézt globální optimum se 100 % úspěšností. Z předchozích grafů absolutní úspěšnosti algoritmu je patrné, že např. Downhill Simplex měl problémy se splněním této podmínky. Tento algoritmus, který je zejména využíván v optimalizaci funkcí v oboru hodnot reálných čísel, byl modifikován pro použití v rámci optimalizace na diskrétních simulačních modelech a nebylo dopředu zřejmé, jak se zachová. Samozřejmě, že nelze opomenout, že na úspěšnost při hledání globální optima v prohledávaném prostoru má velký vliv průběh účelové funkce v tomto prostoru.

U všech takto vybraných sérií se 100 % úspěšností podle prvního kritéria byl následně zjištěn počet, kolikrát se daná série s konkrétní vybranou hodnotou parametru optimalizačního algoritmu vyskytla u simulačního modelu. Tato hodnota byla následně podělena celkovým počtem provedených sérií s vybranou hodnotou parametru pro konkrétní model bez ohledu na úspěšnost konkrétní série. Tím byla vypočtena procentuální úspěšnost sérií s vybranou hodnotou parametru optimalizačního algoritmu.

Z vypočtených výsledků této analýzy byl následně sestrojen sloupcový graf zvlášť pro každý parametr optimalizačního algoritmu. Z výše hodnoty úspěšnosti v tomto grafu lze usuzovat, jak velký vliv na chování algoritmu má tato hodnota. Pokud jsou hodnoty v grafu přibližně stejné, lze konstatovat, že algoritmus je necitlivý na nastavení parametru v takto vymezeném intervalu, a tudíž lze hodnoty parametru nastavit v tomto intervalu na jakoukoliv hodnotu.

12.2 Nalezení vhodného nastavení všech parametrů algoritmu pro konkrétní simulační modely

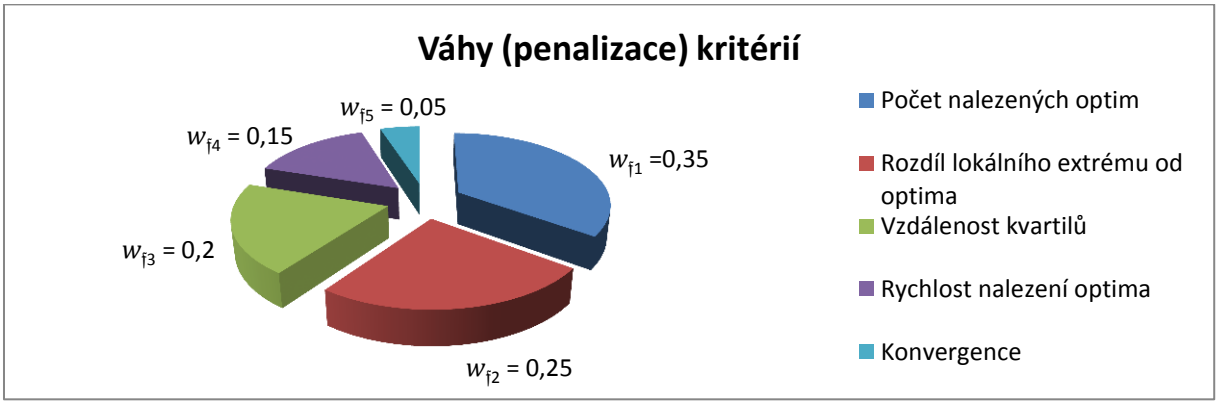
Je důležité zdůraznit, že složením všech jednotlivých vhodných hodnot parametrů algoritmu určených na základě předchozí analýzy, nemusí být získáno ve výsledku ideální nastavení parametrů algoritmu. Optimum jednotlivých částí nemusí být optimem pro celek. Proto byl pro nalezení vhodného nastavení určen další přístup, kterým se určí nejlepší nalezená nastavení podle nalezené nejvyšší hodnoty prvního kritéria, tj. f_1. Pro konkrétní optimalizační algoritmus bude proveden výběr všech sérií, které dosahovaly nejlepší nalezené hodnoty f_1 pro konkrétní simulační model. Jednotlivé takto vybrané série byly ohodnoceny pomocí souhrnné funkce f. Tato souhrnná funkce je definována z důvodu, že uživatel v závislosti na dané situaci může preferovat jisté vlastnosti algoritmu (např. spokojí se i s horším výsledkem, ale potřebuje možné řešení (i když to nebude globální optimum) v co nejkratší době, tzn. upřednostní „Rychlost nalezení optima“ před „Rozdílem lokálního extrému od optima“ potažmo „Vzdáleností kvartilů“). Souhrnná funkce f zahrnuje pět kritérií, která jsou ohodnocena vahami (viz (12.1)).

$$f = w_{f_1} \cdot f_1 + w_{f_2} \cdot f_2 + w_{f_3} \cdot f_3 + w_{f_4} \cdot f_4 + w_{f_5} \cdot f_5 \qquad (12.1)$$

kde:

- f ... Souhrnná funkce preferující jednotlivá kritéria.
- w_{f_1} … Hodnota váhy pro hodnotu kritéria f_1, tj. „Počet nalezených optim nebo hodnot blízkých optimu“, $f_1 \in [0,1]$.
- w_{f_2} … Hodnota váhy pro hodnotu kritéria f_2, tj. „Rozdíl lokálního extrému od optima“, $f_2 \in [0,1]$.
- w_{f_3} … Hodnota váhy pro hodnotu kritéria f_3, tj. „Vzdálenost kvartilů“, $f_3 \in [0,1]$.
- w_{f_4} … Hodnota váhy pro hodnotu kritéria f_4, tj. „Rychlost nalezení optima“, $f_4 \in [0,1]$.
- w_{f_5} … Hodnota váhy pro hodnotu kritéria f_5, tj. „Konvergence“, $f_5 \in [0,1]$.

Je snahou minimalizovat souhrnnou funkci, protože všechny její prvky jsou také minimalizovány (čím menší hodnota tím lepší daný atribut kritéria). Součet vah je roven jedné. Váhy jednotlivých kritérií byly stanoveny podle subjektivních preferencí. Jedná se zejména o preferenci prvních tří kritérií (velikost váhy určuje výši penalizace za nesplnění – je snahou minimalizovat jednotlivá kritéria a tudíž i výslednou hodnotu součtu). Nastavení jednotlivých vah je uvedeno v následujícím grafu (viz Obr. 12-1). Tato funkce bude využita pro nalezení vhodného univerzálního nastavení všech parametrů algoritmu pro všechny vybrané simulační modely.



Obr. 12-1 Stanovené penalizace (váhy) u jednotlivých kritérií

Z předchozího grafu (viz Obr. 12-1) je patrné, že v tomto případě je nejvíce preferováno kritérium vyjadřující počet nalezených optim. U vybraných simulačních modelů byly totiž definovány poměrně malé rozměry prohledávaných prostorů a je tedy žádoucí, aby algoritmus nalezl skoro vždy globální optimum. Bylo také

snahou, aby hranice prohledávaného prostoru u testovaných funkcí byly zvoleny tak, že prohledávaný prostor bude reprezentovat typické hodnoty argumentu účelové funkce.

Po ohodnocení všech vybraných sérií bylo dále provedeno vzestupné setřídění podle hodnoty výsledné ohodnocující funkce. První série v tomto vzestupné žebříčku je tedy nejúspěšnější ze všech provedených sérií z pohledu preferovaných kritérií (hledání globálního optima, rychlosti nalezení optima a konvergence).

12.3 Nalezení vhodného nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Dalším pohledem na algoritmy je univerzalita nastavení parametrů algoritmu. Za účelem nalezení vhodného univerzálního nastavení parametrů algoritmu pro všechny simulační modely byl uplatněn předchozí specifikovaný postup pro nalezení vhodného nastavení parametrů algoritmu na konkrétním simulačním modelu. Do výběru byly zahrnuty všechny série, které splňují podmínku nejvyšší nalezené hodnoty u f_1, v našem případě $f_1 = 0$. Rozdíl, oproti předchozímu postupu je v tom, že je proveden **součet** všech hodnot výsledné ohodnocující funkce u všech simulačních modelů pro konkrétní sérii. Po té je pro všechny takto vybrané série provedeno vzestupné setřídění podle hodnoty součtu výsledné ohodnocující funkce u všech simulačních modelů. Do tabulek bude umístěno pět nejlepších nalezených univerzálních sérií spolu s výslednou hodnotou vyjadřující součet hodnot výsledné ohodnocující funkce u všech simulačních modelů pro konkrétní sérii.

13 Vhodné nastavení parametrů jednotlivých optimalizačních algoritmů na základě provedených sérií

Grafy vyhodnocení jednotlivých optimalizačních algoritmů jsou uvedeny v **příloze disertační práce** vzhledem k jejich velkému počtu a jejich velikosti.

13.1 Random Search

Algoritmus náhodného prohledávání obsahuje pouze jeden parametr, který vyjadřuje to, zda může algoritmus generovat stejné prvky, či nikoliv. Algoritmus je využitelný zejména v případech, kdy nemáme žádnou apriorní znalost o průběhu účelové funkce a jsme schopni provést větší počet simulačních experimentů.

13.1.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

V kapitole bude popsán vliv parametru **Generovat stejné prvky** na chování optimalizačního algoritmu při hledání globálního optima. Do výběru jsou zahrnuty pouze **absolutně úspěšné série** tj. série, které dosáhly 100 % úspěšností podle prvního kritéria.

Ohodnocení jednotlivých hodnot parametrů je provedeno na základě předchozího postupu „Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model“ (viz kapitola 0).

V rámci zachování **stejných** podmínek pro všechny algoritmy bych chtěl také upozornit na fakt, že byly testovány pouze dvě série (viz Tabulka 10-1 Provedené série na simulačních modelech), tudíž úspěšnost nejlepšího nastavení může nabývat hodnoty 0 %, 50 % nebo 100 % vzhledem k provedenému počtu sérií. Z grafu vyjadřujícího úspěšnost sérií s hodnotou parametru **GenerovatStejnePrvky** (viz **příloha** – kapitola 6.1.1 Generovat stejné prvky, Obr. 6-1 Úspěšnost sérií s hodnotou parametru „Generovat stejné prvky“ - Random Search) vyplývá, že optimalizační algoritmus Random Search u simulačních modelů „De Jong, Rosenbrock, MIchalewicz, Ackley, VyrobniLinka“ dosahoval 100 % úspěšnosti (při kritériu $f_1 = 0$) tehdy, pokud byl parametr nastaven na hodnotu **False**, tj. negeneroval stejné prvky.

Chování algoritmu je čistě náhodné, tudíž je velmi závislý na počtu prováděných simulačních experimentů a díky tomu, že u kritéria ukončení byl specifikován maximální počet provedených simulačních experimentů na hodnotu rovnou počtu všech možných řešení v prohledávaném prostoru, dosahoval algoritmus „nadhodnocené“ úspěšnosti.

Tento fakt byl prokázán u simulačního modelu „Penalizace“, kdy prohledávaný prostor obsahoval 53001 možných řešení (prvků v prohledávaném prostoru). U tohoto simulačního modelu tedy byla menší pravděpodobnost nalezení globálního optima. U tohoto simulačního modelu nebyla provedena ani jedna absolutně úspěšná série. Nejlepší provedená série s nastavením „GenerovatStejnePrvky = Pravda“ u tohoto simulačního modelu s velmi členitým průběhem účelové funkce dosahovala $f_1 \cong 0.93$, tzn., cca 7 % úspěšnosti při nalezení globálního optima. S nastavením „GenerovatStejnePrvky = Nepravda“ dosahoval algoritmus Random Search při hledání globálního optima $f_1 \cong 0.97$, tzn., cca 3 % úspěšnosti.

U simulačního modelu „Doprava“ optimalizační algoritmus dosahoval stejné úspěšnosti (cca 47 % úspěšnost), u obou nastavení parametru GenerovatStejnePrvky.

13.2 Downhill Simplex

Algoritmus Downhill Simplex je jednoduchý algoritmus, který pracuje s množinou bodů označovanou jako „Simplex“. Tato množina bodů, obsahuje minimálně $n+1$ bodů, kde n vyjadřuje rozměr prostoru rozhodovacích proměnných. Tyto body jsou nekomplanární (lineárně nezávislé body). Základními fázemi algoritmu jsou **reflexe**, **expanze**, **kontrakce a redukce**. V některých algoritmech jsou využívány jen dvě fáze – reflexe, redukce (viz [14]).

13.2.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Z předchozích grafů průměrné úspěšnosti algoritmů při nalezení optima (viz Obr. 11-1 Průměrná úspěšnost algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu), absolutní úspěšnosti algoritmů při nalezení optima (viz Obr. 11-3 Absolutní úspěšnost sérií algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu) a absolutní neúspěšnosti algoritmů při nalezení optima (viz Obr. 11-8 Absolutní neúspěšnost algoritmů při nalezení optima) je patrné, že algoritmus Downhill Simplex nebyl úspěšný při hledání optima. Dokonce ani v jednom případě ze všech 24 200 různých testovaných sérií u těžších průběhů účelové funkce (simulační modely „Ackley, Doprava, Penalizace“) nedošlo k absolutně úspěšné sérii ($f_1 = 0$), a proto v následujících sloupcových grafech nejsou vykresleny hodnoty.

Tato metoda je užívána zejména při optimalizaci účelových funkcí pracující s rozhodovacími proměnnými, jejichž hodnoty jsou z oboru reálných čísel. Problém algoritmu Downhill Simplex spočívá v jeho podstatě, kdy je využíván bod reflexe, který je vypočten na základě těžiště všech bodů v simplexu. Aby bylo možné určit hodnotu účelové funkce v tomto bodu (prvku) u **diskrétních simulačních modelů**, bylo v našem případě provedeno zaokrouhlení souřadnic směrem k nejbližšímu možnému bodu, určeného krokem u jednotlivé rozhodovací proměnné. Díky tomu dochází k mírnému (v závislosti na velikosti rastru prohledávaného prostoru) odchýlení původního směru (u reálných čísel tato modifikace nemusí být prováděna). Tato modifikace má za výsledek, že dochází k rychlému stagnování do bodů, které v následujících iteracích nejsou měněny (nedochází k diverzifikaci prvků). Tento problém by mohl být např. řešen takovou modifikací algoritmu, že by byla vypočtena hodnota účelové funkce aproximací hodnot účelové funkce nejbližších bodů. Tím však vzroste časová náročnost výpočtu.

13.2.1.1 Reflexe

Reflexe znamená překlopení nejhoršího prvku (bodu) simplexu přes těžiště simplexu. U algoritmu byly testovány i takové případy, že nedochází k žádné reflexi, respektive bod reflexe je stejný jako těžiště simplexu bez nejhoršího bodu (koeficient reflexe byl nulový). Procentní podíl úspěšných sérií je minimální, což se odráží i v hodnotách úspěšnosti hodnot parametru „Reflexe“ (viz **příloha** – kapitola 6.2.1 Reflexe, Obr. 6-2 Úspěšnost sérií s hodnotou parametru „Reflexe“ – Downhill Simplex). Nízká procentní úspěšnost v grafu je zapříčiněna podělením malého počtu absolutně úspěšných sérií vůči velkému počtu všech testovaných sérií.

V následující tabulce (viz Tabulka 13-1) jsou uvedeny doporučené hodnoty u parametru Reflexe. Doporučené hodnoty v závorkách s hvězdičkou jsou uvedeny v případě, kdy nebyla provedena ani jedna absolutně úspěšná série. Pro úplnost tedy doplním hodnoty úspěšnosti nejlepší nalezené série u jednotlivých simulačních modelů: „Ackley“: úspěšnost 60 %, „Doprava“: úspěšnost 80 %, „Penalizace“: úspěšnost 40 %.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Reflexe	[0.6,1.0] Krok 0.2	[1.0,1.4] Krok 0.2	[0.8,1.2] Krok 0.2	$*\{0.6,1.6,1.8\}$	$*\{0.8,1.8\}$	$*$ [0.8,1.6] Krok 0.2	$[0.6,1.2] \cup \{2.0\}$ Krok 0.2

Tabulka 13-1 Doporučené hodnoty u parametru „Reflexe“ - Downhill Simplex

13.2.1.2 Expanze

Koeficient **Expanze** určuje, jak daleko bude překlopen bod reflexe simplexu ve směru, který prochází spojnicí nejhoršího bodu a těžiště. Toto překlápění se děje pouze tehdy, pokud hodnota účelové funkce v bodu reflexe dosáhla lepší hodnoty než dosud nejlepší bod simplexu. Znamená to tedy, že byl nalezen slibný směr, ve kterém se vydat při hledání optima. V grafu úspěšnosti (viz **příloha** - kapitola 6.2.2 Expanze, Obr. 6-3 Úspěšnost sérií s hodnotou parametru „Expanze“ - Downhill Simplex) jsou zachyceny hodnoty u nejlepších sérií ze všech

testovaných sérií. Na základě předchozího grafu jsou v tabulce (viz Tabulka 13-2) uvedeny doporučené hodnoty u parametru Expanze.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Expanze	[0.6,1.2] Krok 0.2	[1.0,1.4] Krok 0.2	[0.4,1.8] Krok 0.2	$*\{1.4,1.6,2.0\}$	$*\begin{Bmatrix}0.0,0.8,\\1.2,1.6\end{Bmatrix}$	$*\{1.6\}$	[0.0,2.0] Krok 0.2

Tabulka 13-2 Doporučené hodnoty u parametru „Expanze“ - Downhill Simplex

13.2.1.3 Kontrakce

V případě, že získaný bod pomocí reflexe není lepší (podle hodnoty účelové funkce) než nejhorší bod simplexu, simplex se zkracuje do nového prvku pomocí kontrakce – prvek je umístěn mezi bod reflexe a těžiště simplexu**.** Koeficient **Kontrakce** určuje vzdálenost umístění nového prvku kontrakce mezi bod reflexe a těžiště simplexu. Hodnoty koeficientu - parametru Kontrakce z nejúspěšnějších testovaných sérií u jednotlivých simulačních modelů jsou zobrazeny v grafu úspěšnosti (viz **příloha** – kapitola 6.2.3 Kontrakce, Obr. 6-4 Úspěšnost sérií s hodnotou parametru „Kontrakce“- Downhill Simplex).

Doporučené hodnoty parametru Kontrakce jsou uvedeny v následující tabulce (viz Tabulka 13-3).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Kontrakce	[0.44,0.99] Krok 0.11	[0.33,0.44] Krok 0.11	[0.33,0.66] Krok 0.11	$*\begin{Bmatrix}0.22,0.44,\\0.66\end{Bmatrix}$	$*$ [0.11,0.66] Krok 0.11	$*$ [0.11,0.88] Krok 0.11	$*$ [0.0,0.99] Krok 0.11

Tabulka 13-3 Doporučené hodnoty u parametru „Kontrakce“- Downhill Simplex

13.2.1.4 Redukce

Koeficient **Redukce** určuje velikost smrštění všech bodů simplexu směrem k nejlepšímu bodu simplexu. Následující tabulka (viz Tabulka 13-4) obsahuje nejlepší hodnoty koeficientu Redukce z absolutně úspěšných série umístěných ve sloupcovém grafu úspěšnosti (viz **příloha** – kapitola 6.2.4 Redukce, Obr. 6-5 Úspěšnost sérií s hodnotou parametru „Redukce“- Downhill Simplex).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Redukce	[0.44,0.88] Krok 0.11	[0.77,0.88] Krok 0.11	[0.22,0.88] Krok 0.11	$*\begin{Bmatrix}0.22,0.33,\\0.66\end{Bmatrix}$	$*\begin{Bmatrix}0.22,0.33,\\0.66,0.77\end{Bmatrix}$	$*$ [0.0,0.99] Krok 0.11	$*$ [0.0,0.99] Krok 0.11

Tabulka 13-4 Doporučené hodnoty u parametru „Redukce“- Downhill Simplex

13.2.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

Pokud srovnáme průměrné hodnoty úspěšnosti (viz Obr. 11-1 Průměrná úspěšnost algoritmu při nalezení optima pro všechny simulační modely a všechny nastavené parametry algoritmu), je jasné že použitá podoba algoritmu pro účely simulačních optimalizace prováděné **na diskrétních simulačních modelech** je spolu s Diferenciální evolucí **malá** – absolutní úspěšnost sérií je 0.17 %, což je oproti Evoluční strategii s hodnotou absolutní úspěšnosti 44 % hodnota neuspokojivá. Pokud se zaměříme na hodnotu f_3 (viz Obr. 11-12 Průměr z hodnot f_3 (vzdálenost kvartilů) u všech sérií kromě absolutně úspěšných sérií pro všechny simulační modely) zjistíme, že největší rozpětí hodnot účelové funkce u poskytovaných výsledků měl algoritmus u simulačního modelu „Michalewicz“ a „Ackley“.

Následující tabulka (viz Tabulka 13-5) obsahuje nalezená „nejlepší“ nastavení všech parametrů algoritmu Downhill Simplex pro konkrétní simulační modely, tj. hodnoty parametrů algoritmu z nejlepších nalezených nastavení setříděných podle souhrnného funkce f, tj., preference různých kritérií pomocí vah.

Název modelu	Reflexe	Expanze	Kontrakce	Redukce
DeJong	0.6	1	0.77	0.88
Michalewicz	1	1.4	0.44	0.88
Rosenbrock	1	0.8	0.44	0.44
Ackley	0.6	1.4	0.66	0.66
Doprava	0.8	1.6	0.55	0.22
Penalizace	1.2	0.6	0.44	0.22
VyrobniLinka	0.4	0.4	0.88	0.88

Tabulka 13-5 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Downhill Simplex

13.2.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Pro úplnost uvádíme následující tabulku, v níž zjistíme, že při porovnání s tabulkou vyjadřující vhodné nastavení parametru Kontrakce (viz Tabulka 13-3 Doporučené hodnoty u parametru „Kontrakce“- Downhill Simplex), je nejčastější hodnotou „Kontrakce = 0.44“.

Reflexe	Expanze	Kontrakce	Redukce	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
1	0.8	0.44	0.77	0.544021023
1.2	1	0.44	0.44	0.567062933
0.6	0.6	0.66	0.55	0.569106865
1.2	1.2	0.44	0.44	0.575461896
0.8	1.8	0.55	0.55	0.580357947

Tabulka 13-6 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Downhill Simplex

13.3 Hill Climbing

Horolezecký algoritmus je založený na „pseudogradientním“ prohledávání povrchu účelové funkce. Dá se tedy předpokládat, že algoritmus bude selhávat u průběhů účelové funkce, které budou multimodální a příliš členité (předčasná konvergence). To se potvrdilo u simulačního modelu „Penalizace“, kde optimalizační algoritmus Hill Climbing selhal. Naopak byl úspěšný u lehkých průběhů účelové funkce zejména z pohledu potřeného počtu simulačních experimentů při hledání globálního optima.

13.3.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Na základě následující analýzy bylo zjištěno, že důležitým parametrem ovlivňujícím efektivitu algoritmu při hledání globální optima je **rozptyl**.

Výše úspěšnosti při hledání globálního optima u vybraných sérií s konkrétní hodnotu parametru optimalizačního algoritmu provedených na konkrétním simulačním modelu (série musí splňovat podmínku, že výše její neúspěšnosti podle prvního kritéria je 0 %, tzn., $f_1 = 0$), je zobrazena pomocí jednoho sloupce v grafu **úspěšnosti**. Výše hodnoty úspěšnosti vyjadřuje procentuální zastoupení absolutně úspěšných sérií s konkrétní hodnotou parametru vůči všem provedeným sériím optimalizačního algoritmu na konkrétním simulačním modelu. Na základě rozdílu výše úspěšností jednotlivých sloupců lze určit vliv hodnoty parametru na úspěšnost optimalizačního algoritmu při hledání optima. Pokud budou jednotlivé sloupce přibližně stejně vysoké (budou dosahovat přibližně stejné úspěšnosti), lze také usuzovat na jistou univerzalitu v nastavení parametru optimalizačního algoritmu.

V použitém modifikovaném algoritmu Hill Climbing bylo namísto generování jediného prvku v okolí nejlepšího bodu z předchozí populace generováno více prvků proto, aby bylo bráněno předčasné konvergenci algoritmu.

Z výsledků úspěšnosti u absolutně úspěšných sérií uvedených v grafu (viz **příloha** – kapitola 6.3.2 Velikost populace, Obr. 6-8 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Hill Climbing) vyplývá, že série s **velikostí populace** jedna dosahují minimálního úspěchu při hledání optima. Z tohoto faktu vyplývá a vizualizace optimalizačních experimentů to také potvrdila, že modifikace základního algoritmu pomocí generování celé skupiny prvků, namísto jediného prvku, byla velmi efektivní a vedla ke snížení rizika předčasné konvergence.

13.3.1.1 Rozptyl

Parametr **Rozptyl** vymezuje hranici oblasti kolem nejlepšího řešení z předchozí populace. V tomto sousedství okolo nejlepšího prvku (jedince) z předchozí skupiny prvků (populace) jsou generovány prvky umístěné do nové skupiny prvků (aktuální populace) na základě stejného náhodného rozdělení pro všechny rozhodovací proměnné simulačního modelu. S růstem hodnoty parametru se zvyšuje **diverzita** generovaných prvků, tedy možných řešení optimalizačního problému.

Z grafu úspěšnosti (viz **příloha** – kapitola 6.3.1 Rozptyl, Obr. 6-6 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Hill Climbing) je patrné, že u lehkých průběhů účelové funkce (nečlenité, konvexní, atd.) lze doporučit menší hodnoty tohoto parametru algoritmu. Z grafu je patrné, že efektivita nalezení globálního optima klesá s použitím větších rozptylů v rámci prohledávaného prostoru u vybraného simulačního modelu. Je to dáno tím, že v sousedství mohou být generovány i horší prvky, než aktuální prvek a v kombinaci s nastavením malého počtu generovaných prvků v sousedství, ze kterých je vybrán nejlepší, dochází k odchýlení od správného směru hledání globálního optima.

Naproti tomu v případě multimodální „Ackleyho“ funkce je úspěšnost u absolutně úspěšných sérií, obsahující malé hodnoty rozptylu, nulová. Je to tím, že díky malému rozptylu, jsou generována taková řešení, která se nemohou vymanit z hlubokých údolí lokálních minim.

V grafu úspěšnosti sérií s hodnotou parametru Rozptyl (viz **příloha** - kapitola 6.3.1 Rozptyl, Obr. 6-6 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Hill Climbing) nejsou zobrazeny žádné sloupce, týkající se simulačního modelu „Penalizace“. Průběh účelové funkce je v tomto případě velmi členitý (viz **příloha** - kapitola 6.3.1 Rozptyl, Obr. 6-7 Horní pohled na povrch účelové funkce simulačního modelu „Penalizace“). Umístění výchozího bodu (počátečního přípustného řešení) bylo na nejvyšším vrcholu účelové funkce. V blízkém okolí počátečního přípustného řešení se účelová funkce svažuje a díky tomu algoritmus konverguje do oblasti lokálního extrému.

Protože algoritmus Hill Climbing byl při optimalizaci simulačního modelu „Penalizace“ neúspěšný, bylo provedeno dodatečné testování sérií s velkými hodnotami rozptylu (viz Tabulka 13-7).

Parametr algoritmu	Krok	Meze		Počet opakování v sérii
		Dolní mez	Horní mez	
Rozptyl	10	10	60	30
VelikostPopulace	10	10	20	
PrvniPopulaceNahodne	1	1	1	

Tabulka 13-7 Testované hodnoty parametrů - Hill Climbing

Na základě výsledků poskytnutých optimalizačním algoritmem bylo zjištěno, že nejlépe ze všech provedených sérií, vzhledem k počtu nalezených optim, dopadly série kdy „Rozptyl = 10“ a „VelikostPopulace = 20“ a série kdy „Rozptyl = 10“ a „VelikostPopulace = 10“. Pouze tyto dvě série dosahovaly při hledání globálního optima v prohledávaném prostoru cca 47 % a 33 % úspěšnosti. Všechny ostatní série dosahovaly úspěšnosti při nalezení globálního minima menší než 18 %.

Nejhůře, co do počtu nalezených globálních optim, dopadly série s nejvyšším rozptylem spolu s nejvyšší velikostí populace. Je důležité ale také zdůraznit, že pokud bychom měli brát v úvahu vzdálenost nalezených řešení od globálního optima, tato dvě předchozí nastavení nejsou tak vhodná.

Na základě hodnocení úspěšnosti uvedeného v grafu a další analýzy, která byla provedena pomocí seřazení sérií podle souhrnné funkce f a dalšího testování parametrů (s vyšším počtem optimalizačních experimentů v sérii) jsou doporučeny hodnoty parametru **Rozptyl** pro jednotlivé simulační modely, které jsou uvedeny v následující tabulce (viz Tabulka 13-8).

Poznámka: Hodnoty parametru uvedené ve **složené** závorce jsou výčtem hodnot. Hodnoty parametru uvedené v **hranaté** závorce jsou intervaly hodnot. V **závorce s hvězdičkou** jsou uvedeny hodnoty parametru optimalizačního algoritmu u sérií, které nebyly absolutně úspěšné, tzn., že nesplňovaly podmínku pro výběr, která byla specifikována pro všechny simulační modely ($f_1 = 0$). Tyto nejlepší série ($f_1 \neq 0$) dosáhly nejvyšší úspěšnosti vůči všem ostatním provedeným sériím na simulačním modelu.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Rozptyl	{1,3,5,} {7,9}	{1,3,5,7}	{1,3,5,7}	{11,13,15,} {21,23,25}	{3,5}	∗ {9,19}	{1,3,5}

Tabulka 13-8 Doporučené hodnoty u parametru „Rozptyl“ - algoritmus Hill Climbing

13.3.1.2 Velikost populace

Na základě grafu úspěšnosti sérií s konkrétními hodnotami parametru **VelikostPopulace** (viz **příloha** – kapitola 6.3.2 Velikost populace, Obr. 6-8 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Hill Climbing) lze tvrdit, že díky implementaci principu generování prvků v celých populacích, namísto jednoho prvku, je u základního algoritmu Hill Climbing zvýšena účinnost algoritmu při hledání globálního optima. Standartní Hill Climbing totiž využívá pouze generování jediného prvku v okolí nejlepšího bodu. Série obsahující pouze jediný prvek předčasně konvergovaly u všech simulačních modelů. U lehkých průběhů účelové funkce převládá v úspěšnosti vyšší hodnota velikosti populace. V sousedství je díky generování většího počtu prvků nalezen vhodný směr nejprudšího spádu – přiblížení se ke gradientu, kterým se má algoritmus v další populaci vydat. Pokud je v sousedství generováno menší množství prvků, algoritmus může a také díky náhodnému rozdělení generuje jako nová možná řešení i nekvalitní prvky. Protože se algoritmus vždy musí přesunout do nejlepšího nalezeného prvku z minulé populace, může uhnout mimo směr největšího spádu.

Na základě hodnocení uvedeného v grafu (viz **příloha** - kapitola 6.3.2 Velikost populace, Obr. 6-8 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Hill Climbing) a další analýzy, která byla provedena pomocí seřazení sérií podle souhrnné funkce f a dalšího testování parametrů (s vyšším počtem optimalizačních experimentů v sérii), jsou doporučeny hodnoty parametru **VelikostPopulace** pro jednotlivé simulační modely, které jsou uvedeny v následující tabulce (viz Tabulka 13-9).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Velikost populace	[3,27] Krok 2	[3,27] Krok 2	[3,27] Krok 2	[17,27] Krok 2	[3,27] Krok 2	∗ {9,11,25}	[3,27] Krok 2

Tabulka 13-9 Doporučené hodnoty u parametru „Velikost populace“ - algoritmus Hill Climbing

13.3.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

Jak již bylo řečeno, použitím všech jednotlivých doporučených hodnot parametrů algoritmu (určených na základě předchozí analýzy pro každý parametr odděleně), nemusí být ve výsledku ideální celkové nastavení algoritmu. Na základě specifikovaného přístupu - „Nalezení vhodného nastavení všech parametrů algoritmu pro konkrétní simulační modely“ (viz kapitola 0), byl proveden výběr sérií s nejvyšší úspěšností podle kritéria f_1 a

dále bylo provedeno vzestupné setřídění všech sérií podle hodnot získané na základě souhrnné funkce ƒ (preference jednotlivých kritérií).

V následující tabulce (viz Tabulka 13-10) jsou uvedena vhodná nastavení pro konkrétní simulační modely u optimalizačního algoritmu Hill Climbing. Chování algoritmu Hill Climbing je zejména závislé na parametru Rozptyl. Pro tento parametr je vhodné u lehkých průběhů účelové funkce („De Jong, Michalewicz, Rosenbrock, Doprava, VyrobniLinka“) použít menší hodnoty rozptylu. Naproti tomu u členitějších průběhů funkce („Ackley, Penalizace“) je vhodné použít větší hodnotu rozptylu i větší velikost populace.

Název modelu	Rozptyl	Velikost populace
DeJong	5	7
Michalewicz	3	3
Rosenbrock	3	5
Ackley	21	19
Doprava	3	7
Penalizace	9	11
VyrobniLinka	1	5

Tabulka 13-10 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely – Hill Climbing

13.3.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Z následující tabulky, v níž je umístěno 5 vhodných (nejlepších) nastavení všech parametrů algoritmu ze **všech** testovaných nastavení, která byla prováděna pro **všechny** vybrané simulační modely u algoritmu Hill Climbing vyplývá, že důležitým faktorem je velikost rozptylu. U rozptylu je ve výběru nejčetnější hodnota 5. Pokud srovnáme hodnoty velikosti populace, jsou různorodé. Ze sloupce s hodnotou souhrnné funkce ƒ (zahrnující hodnocení pěti kritérií) u všech simulačních modelů je patrné, že hodnoty nejsou tolik rozdílné, tudíž chování algoritmu je pro testované modely dosti podobné.

Rozptyl	Velikost populace	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
7	7	0.663372123
5	3	0.725217034
5	9	0.74561076
5	21	0.752476908
5	5	0.794562717

Tabulka 13-11 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Hill Climbing

Pro algoritmy založené na principu gradientních metod je velmi problémové uniknout z lokálních extrémů. Tento nedostatek by mohl být oslaben použitím náhodných restartů – algoritmus bude několikrát spuštěn z různých náhodně vygenerovaných počátečních řešení.

13.4 Tabu Search

Algoritmus Tabu Search – algoritmus zakázaného prohledávání, je modifikací předchozího algoritmu Hill Climbing. U algoritmu Hill Climbing může dojít k zacyklení, kdy se algoritmus po určitém počtu kroků vrací k lokálnímu extrému, ve kterém již byl. Použitím krátkodobé paměti již navštívených prvků, které se nemohou opakovaně používat, se algoritmus snaží předejít uvíznutí v lokálním extrému.

13.4.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Na základě provedené analýzy parametrů optimalizační algoritmu Tabu Search lze konstatovat, že největší vliv na úspěšnost série má parametr určující velikost **rozptylu**. To se dá předpokládat i díky implementaci shodné podstaty s algoritmem Hill Climbing. Dalším podstatným vlivem na úspěšnost sérií působí parametr **velikost populace**. Na základě analýzy lze pro všechny simulační modely doporučit generovat nové prvky na základě dosud nalezeného nejlepšího prvku oproti možnosti generování prvků na základě nejlepšího prvku z předchozí populace. Podle výsledků provedených testů s různou **délkou zakázaného seznamu** („Tabu Length“) lze také konstatovat, že různé velikosti seznamu zakázaných prvků měly minimální vliv na chování algoritmu na vybraných simulačních modelech.

13.4.1.1 Rozptyl

V dalším grafu úspěšnosti (viz **příloha** – kapitola 6.4.1 Rozptyl, Obr. 6-9 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Tabu Search) je patrný princip vzrůstajícího a klesajícího trendu úspěšnosti u absolutně úspěšných sérií s hodnotami **rozptylu**. V porovnání s předchozím grafem úspěšnosti sérií s konkrétní hodnotou parametru rozptyl u algoritmu Hill Climbing, je patrný nárůst úspěšnosti algoritmu Tabu Search u simulačního modelu „Ackley“ a „Doprava“. U provedených sérií algoritmu Tabu Search, kde „Rozptyl = 1“, je zřetelný propad úspěšnosti u všech simulačních modelů. V následující tabulce jsou uvedeny doporučené hodnoty rozptylu pro jednotlivé simulační modely (viz Tabulka 13-12).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Rozptyl	[3,11] Krok 2	[3,11] Krok 2	[3,15] Krok 2	[11,15] Krok 2	[3,7] Krok 2	{7,13}	[1,9] Krok 2

Tabulka 13-12 Doporučené hodnoty u parametru „Rozptyl“ - Tabu Search

U Tabu Search lze z grafů úspěšnosti nastavení usoudit, že u tohoto optimalizačního algoritmu velice záleží na nastavení hodnot rozptylu, zejména u lehkých průběhů účelové funkce. Z grafu úspěšnosti sérií s hodnotou parametru „Rozptyl“ optimalizačního algoritmu Tabu Search (viz **příloha**) je patrné, že série s konkrétní mezní hodnotou rozptylu mohou u lehkých průběhů účelové funkce dosahovat až 96 % úspěšnosti.

13.4.1.2 Velikost populace

U simulačních modelů se dá obecně předpokládat, že nastavení „Velikost populace = 1“ povede k předčasné konvergenci. To se potvrdilo i v případech testovaných simulačních modelů, kde se sice algoritmus díky implementaci seznamu zakázaných prvků s jediným prvkem chová lépe než algoritmus Hill Climbing, ale úspěšnost sérií s touto hodnotou parametru je menší, než u sérií s větší velikostí populace (viz **příloha** – kapitola 6.4.2 Velikost populace, Obr. 6-10 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Tabu Search). U nepříliš členitých povrchů účelové funkce („De Jong“, „Michalewicz“, „Rosenbrock“, „Doprava“, „VyrobniLinka“) je patrná velká úspěšnost sérií pokud byla zvolena menší velikost populace. V následující tabulce jsou doporučené hodnoty parametru **Velikost populace** u algoritmu Tabu Search (viz Tabulka 13-13).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Velikost populace	[3,13] Krok 2	[3,11] Krok 2	[3,19] Krok 2	[3,25] Krok 2	[3,15] Krok 2	{11}	[1,9] Krok 2

Tabulka 13-13 Doporučené hodnoty u parametru „Velikost populace“ - algoritmus Tabu Search

13.4.1.3 Délka zakázaného seznamu

Parametr **Tabu Length**, který vyjadřuje délku zakázaného seznamu, určuje počet prvků, které nesmí být znovu testovány, tzn., pokud je vygenerován shodný prvek, tento prvek je zamítnut. Z tohoto seznamu jsou staré prvky postupně vymazány a nahrazeny nově vygenerovanými prvky – metoda FIFO. Z grafu úspěšností sérií s hodnotami parametru Tabu Length (viz **příloha** – kapitola 6.4.3 Délka zakázaného seznamu, Obr. 6-11 Úspěšnost sérií s hodnotou parametru „Tabu Length“ - Tabu Search) je patrné, že rozdíly mezi hodnotami různých sloupců jsou na první pohled minimální, což by mohlo vést k závěru, že hodnota tohoto parametru má minimální vliv na úspěšnost algoritmu.

Pro potvrzení nebo vyvrácení hypotézy minimálního vlivu výše hodnoty tohoto parametru na chování algoritmu Tabu Search byly provedeny další série, v nichž byly testovány vyšší hodnoty parametru Tabu Length. Hodnoty ostatních parametrů optimalizačního algoritmu Tabu Search v těchto sériích byly určeny na základě předchozích pokusů provedených za účelem nalezení vhodného nastavení algoritmu pro simulační modely. Hodnoty parametrů algoritmu jsou uvedeny v následující tabulce (viz Tabulka 13-14).

Parametr algoritmu	Krok	Meze		Série – počet opakování
		Dolní mez	Horní mez	
PrvniPopulaceNahodne	1	1	1	30
VelikostPopulace	2	3	5	
Rozptyl	2	5	7	
TabuLength	10	60	100	
OptimalizacePodlePosledníPopulace	1	0	1	

Tabulka 13-14 Provedené série dalších hodnot parametru „Tabu Length“ na všech simulačních modelech

Z výsledků dodatečné analýzy vlivu délky zakázaného seznamu vyplynulo, že v případě jednoduchých průběhů účelové funkce („De Jong, Rosenbrock“) vyšší hodnoty parametru Tabu Length nemají žádný vliv na úspěšnost chování algoritmu Tabu Search – hodnoty úspěšnosti byly totožné. Souhrnně by se dalo říci, že parametr Tabu Length má minimální vliv na úspěšnost algoritmu při hledání globálního optima, protože odchylky hodnot úspěšnosti jednotlivých sloupců u provedených sérií byly minimální.

Při porovnání jednotlivých sloupců muselo být bráno v potaz, že hodnoty úspěšnosti jsou v tomto případě přímo úměrné počtu provedených sérií, kterých v tomto případě bylo provedeno 280. V dodatečně provedených experimentech u simulačního modelu „Penalizace“ se nevyskytla ani jedna absolutně úspěšná série. Na tomto základě lze znovu usoudit, že parametr Tabu Length má u vybraných simulačních modelů minimální vliv na úspěšnost při hledání globálního extrému.

V následující tabulce (viz Tabulka 13-15) jsou uvedeny doporučené hodnoty délky zakázaného seznamu. V závorce s hvězdičkou je uvedena doporučená hodnota ze sérií u simulačního modelu „Penalizace“, které nebyly absolutně úspěšné, ale byly nejlepší vůči ostatním provedeným sériím na simulačním modelu.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Tabu Length	[10,100] Krok 10	[10,100] Krok 10	[10,100] Krok 10	[10,50] ∪ [80,100] Krok 10	[10,90] Krok 10	{20,30}	[20,50] ∪ [70,100] Krok 10

Tabulka 13-15 Doporučené hodnoty u parametru „Tabu Length“ - Tabu Search

13.4.1.4 Optimalizace podle poslední populace

Parametr **Optimalizace podle poslední populace** datového typu Boolean určuje, zda další generovaná populace bude vytvořena na základě nejlepšího prvku v poslední populaci nebo bude generována na základě dosud nejlepšího nalezeného prvku ze všech předchozích populací. Pokud bude populace generována na základě poslední populace, má algoritmus možnost uniknout z lokálních extrémů, protože není vázán na dosud nejlepší prvek, který byl nalezen. Tento parametr nahrazuje rozdíl mezi algoritmem Local Search – lokální prohledávání (populace je generována podle dosud nejlepšího nalezeného prvku) a algoritmem Hill Climbing (populace je generována podle nejlepšího prvku poslední populace). Otázkou tedy je, zda má tento parametr významný vliv na úspěšnost algoritmu u složitějších a příliš členitých povrchů účelové funkce. Z grafu (viz **příloha** – kapitola 6.4.4 Optimalizace podle poslední populace, Obr. 6-12 Úspěšnost sérií s hodnotou parametru „Optimalizace podle poslední populace“ - Tabu Search) je patrné, že je u algoritmu lepší preferovat generování nových prvků na základě dosud nalezeného nejlepšího prvku ze všech předchozích populací namísto generování nových prvků na základě nejlepšího prvku z předchozí populace.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Optimalizace podle poslední populace	Pravda	Pravda	Pravda	Pravda	Pravda	Nepravda	Pravda

Tabulka 13-16 Doporučené hodnoty u parametru „Optimalizace podle poslední populace“ - Tabu Search

13.4.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

Z tabulky nalezených vhodných nastavení všech parametrů algoritmů (viz Tabulka 13-17) lze usoudit, že u algoritmu Tabu Search je vhodné volit u lehkých průběhů účelové funkce menší hodnoty parametru Rozptyl a menší velikost populace. V případě multimodální „Ackleyho“ funkce je lepší volit hodnoty rozptylu vyšší. Je to dáno zejména tím, že s vyšší pravděpodobností dochází ke generování prvku, který unikne z údolí vzdálenějšího lokálního extrému od globálního optima do údolí dalšího lokálního extrému, blíž ke globálnímu optimu.

Název modelu	Rozptyl	Velikost populace	Délka zakázaného seznamu	Optimalizace podle poslední populace
DeJong	3	3	50	Pravda
Michalewicz	7	3	50	Pravda
Rosenbrock	7	11	50	Pravda
Ackley	11	5	10	Nepravda
Doprava	5	15	50	Pravda
Penalizace	13	11	10	Pravda
VyrobniLinka	5	7	40	Pravda

Tabulka 13-17 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Tabu Search

13.4.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Do následující tabulky (viz Tabulka 13-18) je umístěno 5 nalezených vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely. Z tabulky je patrné, že je vhodné u vybraných simulačních modelů volit menší hodnotu rozptylu. Jedná se zejména o jednodušší průběhy účelové funkce.

Rozptyl	Velikost populace	Délka zakázaného seznamu	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
5	15	50	0.007208159
5	13	40	0.008635924
5	13	50	0.008897034
5	15	20	0.013545917
5	15	40	0.01421574

Tabulka 13-18 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Tabu Search

13.5 Simulated Annealing

Algoritmus simulovaného žíhání vychází z fyzikální podstaty žíhání prováděného pro odstranění defektů krystalické mřížky. Každému krystalu (reprezentovaný přípustným řešením) je přiřazena energie ve formě hodnoty účelové funkce. Teplota krystalu je reprezentována parametrem teplota. Riziku předčasné konvergence se algoritmus snaží bránit prováděním velkých změn v počátku optimalizačního experimentu. Velikost změny tedy závisí na teplotě. Čím vyšší je teplota, tím větší se provádí změny. Generování nového prvku se provádí stejným principem jako u pseudogradientních metod. Horší prvek je přijat jen s jistou pravděpodobností, která je závislá na teplotě. Teplota je při přijetí horšího prvku snižována (v algoritmu simulovaného žíhání je implementována možnost snižovat teplotu tehdy, pokud je náhodně vygenerované číslo menší než pravděpodobnost přijetí horšího řešení nebo teplota bude snižována vždy, pokud bude vygenerován horší prvek). Pokud dojde ke snížení teploty pod minimální hranici teploty, teplota je opětovně zvýšena na počáteční specifikovanou hodnotu. K ukončení optimalizačního experimentu dochází v případě splnění některého z definovaných kritérií ukončení.

13.5.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Na základě provedené analýzy parametrů optimalizačního algoritmu Simulated Annealing lze konstatovat, že největší vliv na úspěšnost série má parametr **minimální teplota**. Hodnota parametru určuje hranici teploty, kdy je teplota opět nastavena na počáteční hodnotu. Dalším důležitým parametrem je varianta **generovat dle rozptylu** (generování prvků v sousedství nebo na celém intervalu rozhodovací proměnné) a s tím související výše **rozptylu** (počet kroků vymezující sousedství).

13.5.1.1 Rozptyl

Parametr **rozptyl** má stejnou podstatu jako u algoritmu Hill Climbing nebo Tabu Search, tj., parametr vymezuje velikost oblasti pro jednotlivé rozhodovací proměnných předchozího řešení, v níž bude generován nový prvek na základě stejného náhodného rozdělení pro všechny rozhodovací proměnné simulačního modelu. Z výsledků grafu (viz **příloha** – kapitola 6.5.1 Rozptyl, Obr. 6-13 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Simulated Annealing) vyplývá, že u lehčích průběhů účelové funkce („De Jong, Michalewicz, Rosenbrock“) je efektivnější použití menší hodnoty rozptylu na rozdíl od těžších průběhů účelové funkce („Ackley“), kde se osvědčilo použít větší hodnoty rozptylu, tzn., větší pravděpodobnost vygenerování prvku mimo údolí lokálního extrému. V případě lehčích průběhů účelové funkce, kdy je u algoritmu nastavena velká hodnota rozptylu (větší hranice okolí prvku, ve kterém bude generován nový prvek), má algoritmus možnost generovat horší vzdálenější prvky (prvky s horší hodnotou účelové funkce), a protože v počátečním stádiu prohledávání algoritmus s vyšší pravděpodobností akceptuje i tyto horší vzdálenější prvky (přijmutí závisí na teplotě), dochází k tomu, že algoritmus skáče na odlehlejší místa. Každým akceptováním horšího prvku se pravděpodobnost

přijetí horšího prvku snižuje (nebo zůstává stejná, tj. podle toho, jakou variantu optimalizačního algoritmu uživatel zvolil – booleovský parametr „**SnizovatTeplotuJenPriPrijetiHorsihoReseni**“). Uplatněním principu přijetí horšího řešení lze sice zabránit předčasné konvergenci, ale zároveň se u lehčích průběhů účelové funkce vystavujeme tomu, že algoritmus nestihne dosáhnout globálního optima díky předčasnému ukončení, tj., splnění kritéria ukončení, při kterém byl vyčerpán povolený počet provedených kroků.

V následující tabulce (viz Tabulka 13-19) jsou hodnoty doporučených hodnot parametru rozptylu pro algoritmus simulovaného žíhání.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Rozptyl	[1,13] Krok 2	[3,13] Krok 2	[3,13] Krok 2	[1,13] Krok 2	[1,13] Krok 2	1	[1,13] Krok 2

Tabulka 13-19 Doporučené hodnoty u parametru „Rozptyl“ - Simulated Annealing

U simulačního modelu „Penalizace“ došlo pouze k jediné absolutně úspěšné sérii s hodnotou parametru „Rozptyl = 1“. Za účelem zkoumání dalších hodnot parametrů a jejich vlivu na chování optimalizačního algoritmu v případě simulačního modelu „Penalizace“, byly provedeny další série (viz Tabulka 13-20), které v sobě na základě analýzy úspěšnosti jednotlivých parametrů zahrnují slibnější hodnoty těchto parametrů algoritmu.

Parametr algoritmu	Krok	Meze		Počet opakování v sérii
		Dolní mez	Horní mez	
Rozptyl	5	1	26	30
MenitJenJedenParametr	1	0	1	
Beta	0.045	0.01	0.1	
MinimalniTeplota	0.0045	0.001	0.01	
GenerovatDleRozptylu	1	0	1	
SnizovatTeplotuJenPriPrijetiHorsihoReseni	1	0	1	
PrvniPopulaceNahodne	1	1	1	

Tabulka 13-20 Testované hodnoty parametrů - Simulated Annealing

Z předchozí tabulky série (viz Tabulka 13-20) je patrné, že pro parametr rozptylu v případě simulačního modelu „Penalizace“ byly dodatečně testovány hodnoty 1; 6; 11; 16; 21; 26. Nejlepší série, ze všech testovaných sérií, při hledání optima dosáhla úspěšnosti 23.33 %. Této hodnoty úspěšnosti dosáhly pouze dvě série a to s hodnotami rozptylu 16 a 26.

13.5.1.2 Měnit jen jeden parametr

Na základě literatury [16] byla do algoritmu implementována při generování nového prvku podle předchozího prvku možnost měnit jen jednu rozhodovací proměnnou předchozí prvku za účelem získání nového prvku. Znamená to, že po náhodném výběru složky (rozhodovací proměnné) prvku bude měněna pouze hodnota této složky a ostatní zůstanou stejné jako pro předchozí prvek.

Výsledkem uplatnění tohoto principu je pohyb algoritmu pouze v jednom směru (podobnost s metodami založenými na cyklické záměně proměnných). Z výsledků v grafu (viz **příloha** – kapitola 6.5.2 Měnit jen jeden parametr, Obr. 6-14 Úspěšnost sérií s hodnotou parametru „Měnit jen jeden parametr“ - Simulated Annealing) lze konstatovat, že u lehkých průběhů účelové funkce lze doporučit při generování nového řešení (prvku) měnit všechny rozhodovací proměnné předchozího prvku. V případě simulačního modelu „Michalewicz a Ackley“ je vidět, že lepšího výsledku dosahovaly série s nastavením „Měnit jen jeden parametr = Pravda“.

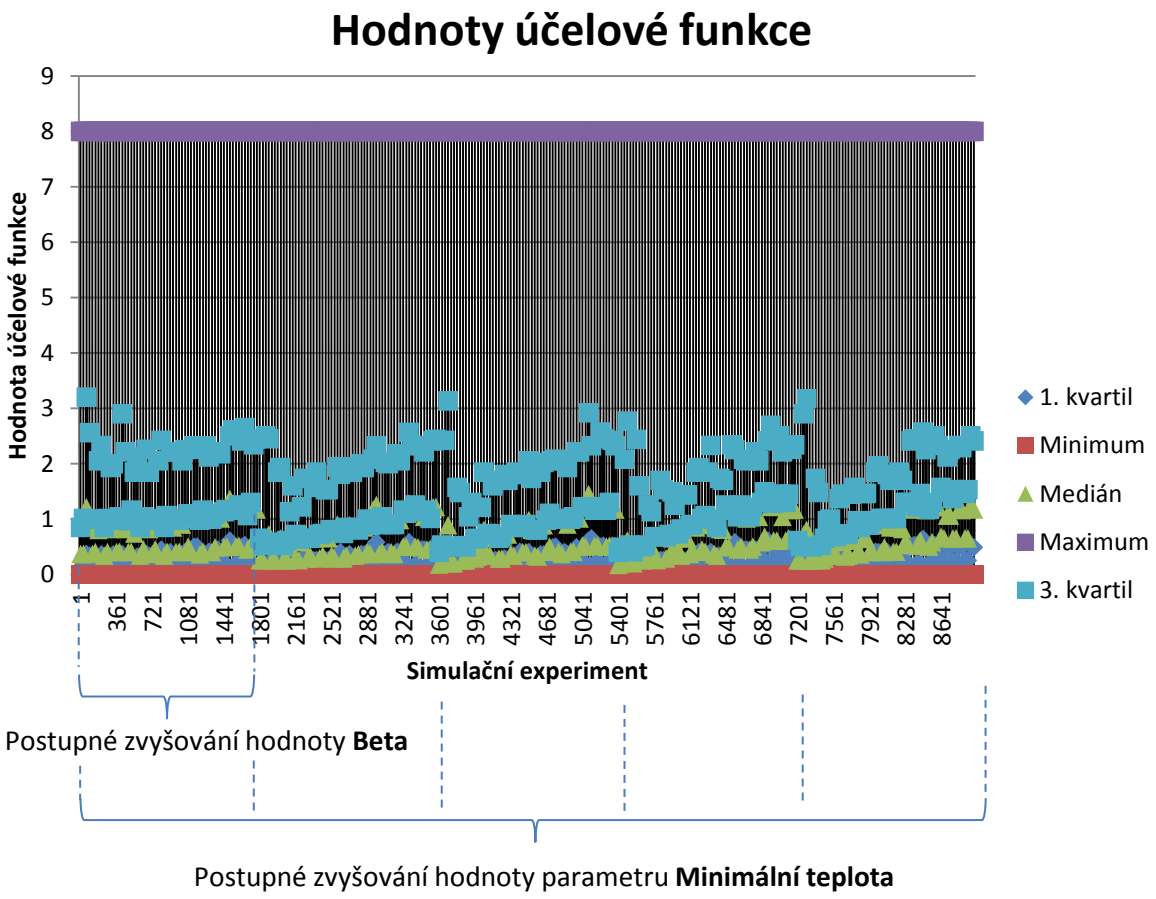
V následující tabulce (viz Tabulka 13-16) jsou hodnoty doporučených hodnot parametru „Měnit jen jeden parametr“ pro algoritmus simulovaného žíhání.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Měnit jen jeden parametr	Nepravda	Pravda	Nepravda	Pravda	Nepravda	Pravda	Nepravda

Tabulka 13-21 Doporučené hodnoty u parametru „Měnit jen jeden parametr“ - Simulated Annealing

13.5.1.3 Beta

Nezáporná konstanta „Beta“ určuje výši poklesu teploty při přijetí horšího řešení ($\beta \geq 0 \wedge \beta \leq 1$). V algoritmu simulovaného žíhání je výše teploty určována podle vzorce pro simulované hašení (viz literatura [16]). V případě posuzování vlivu parametru „Beta“ lze z grafu (viz **příloha** – kapitola 6.5.3 Beta, Obr. 6-15 Úspěšnost sérií s hodnotou parametru „Beta“ - Simulated Annealing) usoudit, že výše hodnoty parametru algoritmu nemá zásadní vliv na úspěšnost při hledání globálního optima (na rozdíl od minimální teploty), ale v převážné většině případů lze doporučit menší hodnotu parametru **Beta**. Díky pomalejšímu snižování teploty totiž algoritmus akceptuje méně horších prvků a dochází tak k menší diverzitě prvků. Tuto domněnku potvrzuje i následující krabicový graf (viz Obr. 13-1) vyjadřující konvergenci pomocí parametru f_5.



Obr. 13-1 Krabicový graf zachycující vliv výše hodnoty parametru „Beta“ a „Minimální teplota“ na hodnoty účelové funkce prvků u provedených sérií na simulačním modelu „De Jong“ - Simulated Annealing

Na základě analýzy krabicového grafu zachycujícího vliv výše uvedené hodnoty parametru **Beta** na hodnoty konvergence f_5 sérií provedených na simulačním modelu „De Jong“ (viz Obr. 13-1) lze konstatovat, že parametr

Beta má v tomto případě menší vliv na diverzitu než parametru **minimální teplota**. Krabicový graf zachycuje konvergenci $\bar{f}_5$ hodnot účelové funkce prvků (rozsah hodnot účelové funkce generovaných prvků během optimalizačního procesu) k hodnotě účelové funkce globálního optima – v tomto případě globálního minima

V krabicovém grafu jsou jasně patrné zvyšující se trendy hodnot účelové funkce mediánu a 3. kvartilu u sérií, jejichž hodnota minimální teploty byla postupně zvyšována (série: 1÷1800; 1801÷3600; 3601÷5400; 5401÷7200; 7201÷9000). Tento trend je větší než u postupného zvyšování parametru Beta. Hodnota maxima je konstantní, protože počátečním přípustným řešením u všech optimalizačních experimentů byl stejný (nejhorší) bod v prohledávaném prostoru, jehož hodnota účelové funkce je hodnota maxima.

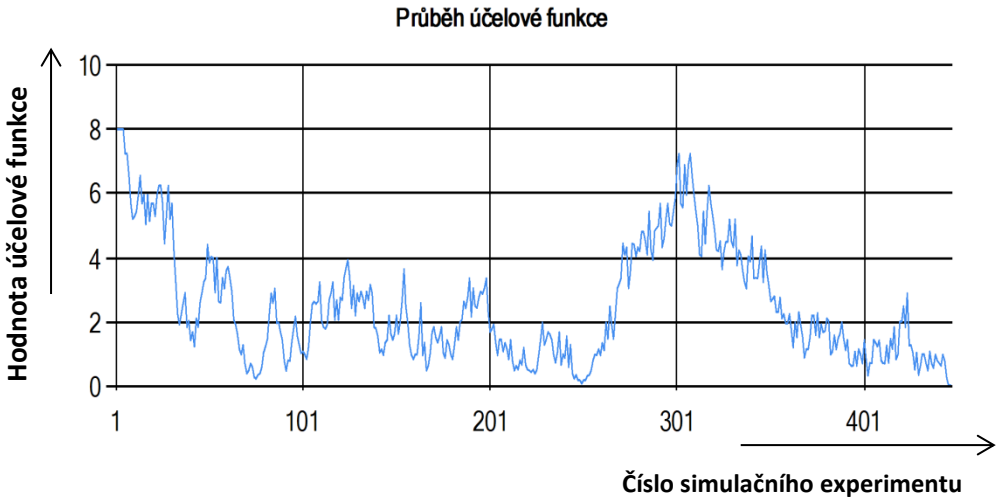
U simulačního modelu „Penalizace“ byl zaznamenána pouze jedna absolutně úspěšná série s hodnotou „Beta = 0.94“. Z toho důvodu nelze usuzovat na vliv této specifické hodnoty parametry na chování algoritmu. V rámci dodatečného testování sérií u simulačního modelu „Penalizace“ (viz Tabulka 13-20) byly testovány nižší hodnoty parametru „Beta“ a to 0.01; 0.055; 0.1. V tomto případě nebyla nalezena ani jedna absolutně úspěšná série. Nejlepší dosažená hodnota úspěšnosti při hledání optima byla 23.33 % a to pouze pro dvě série s hodnotami „Beta = 0.055“ a „Beta = 0.1

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Beta	[0.1,0.94] Krok 0.14	[0.1,0.94] Krok 0.14	[0.1,0.94] Krok 0.14	[0.1,0.94] Krok 0.14	[0.1,0.94] Krok 0.14	0.94	[0.1,0.94] Krok 0.14

Tabulka 13-22 Doporučené hodnoty u parametru „Beta“ - Simulated Annealing

13.5.1.4 Minimální teplota

Pokud se v průběhu optimalizačního experimentu teplota vlivem přijímání horších řešení dostane pod mezní hodnotu parametru (konstanty) nazvaného jako „Minimální teplota“, je teplota opět nastavena na svojí počáteční hodnotu 1 (viz Obr. 13-2), díky čemuž se zvyšuje pravděpodobnost přijetí horšího řešení. Tento krok je v algoritmu implementován zejména z důvodu zabránění předčasné konvergence do lokálního extrému.



Obr. 13-2 Vliv opětovného zvýšení teploty (zvýšení pravděpodobnosti přijetí horšího řešení) na průběh optimalizačního experimentu u simulačního modelu "De Jong"- Simulated Annealing

Z grafu (viz **příloha** – kapitola 6.5.4 Minimální teplota, Obr. 6-16 Úspěšnost sérií s hodnotou parametru „Minimální teplota“ - Simulated Annealing) vyplývá, že skoro u všech modelů (mimo dvou malých odchylek u simulačních modelů „Ackley, MIchalewicz“) převažuje nejnižší testovaná hodnota parametru minimální teploty.

Z výrazně klesajícího trendu úspěšnosti, lze usuzovat, že výše hodnoty tohoto parametru má velký vliv na úspěšnost při hledání optima.

U lehkých průběhů účelové funkce se tento jev dal předpokládat. Díky nastavení parametru minimální teploty na nejnižší hodnotu, algoritmus může provést více iterací, než bude teplota (pravděpodobnost přijetí horšího řešení) opětovně nastavena na počáteční hodnotu (pravděpodobnost přijetí horšího řešení je největší). Tím dochází k pohybu algoritmu v oblastech, ve kterých se nachází prvky s lepší hodnotou účelové funkce. Pokud je nastavena vyšší hodnota minimální teploty, algoritmus po menším počtu kroků nastaví opět „MinimalniTeplota = 1“ (ve snaze uniknout z oblasti lokálního extrému) a tím dochází k putování algoritmu směrem do oblasti prvků s horšími hodnotami účelové funkce.

Skoro u všech simulačních modelů (mimo simulačních modelů „Ackley“ a „Michalewicz“) převažuje nejnižší testovaná hodnota parametru minimální teploty, tj. „MinimalniTeplota = 0.01“. V rámci dodatečného testování tohoto parametru byly na simulačním modelu „Penalizace“ provedeny série s nižšími hodnotami tohoto parametru (viz Tabulka 13-20). Hodnoty parametru MinimalniTeplota byly 0.001; 0.0055; 0.01. Ani v jednom případě nebyla nalezena absolutně úspěšná série. Nejlepší dosažená hodnota úspěšnosti při hledání optima byla 23.33 %. Takové úspěšnosti dosáhly pouze dvě série s hodnotami „MinimalniTeplota = 0.0055“ a „MinimalniTeplota = 0.001“. Z těchto dvou sérií dopadla lépe série s menší hodnotou parametru „MinimalniTeplota“ a to s dvakrát vyšší úspěšnosti než série s vyšší minimální teplotou.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Minimální teplota	[0.01,0.13] Krok 0.04	[0.01,0.09] Krok 0.04	[0.01,0.13] Krok 0.04	[0.01,0.13] Krok 0.04	[0.01,0.09] Krok 0.04	0.01	[0.01,0.25] Krok 0.04

Tabulka 13-23 Doporučené hodnoty u parametru „Minimální teplota“ - Simulated Annealing

13.5.1.5 Generovat dle rozptylu

Některé typy algoritmů simulovaného žíhání (např. dle [16]) využívají při generování nového prvku možnost generovat hodnotu rozhodovací proměnné nového prvku (souřadnici prvku v prohledávaném prostoru) v celém intervalu rozhodovací proměnné. Za účelem srovnání byla v optimalizačním algoritmu implementována možnost zvolit generování hodnot rozhodovacích proměnných nového prvku buď v celém příslušném intervalu rozhodovací proměnné, nebo hodnoty rozhodovacích proměnných jsou generovány na základě rozptylu určujícího velikost okolí předcházejícího prvku, v němž bude generován nový prvek.

Podle grafu (viz **příloha** – kapitola 6.5.5 Generovat dle rozptylu, Obr. 6-17 Úspěšnost sérií s hodnotou parametru „Generovat dle rozptylu“ - Simulated Annealing) je patrné, že u většiny absolutně úspěšných sérií vybraných simulačních modelů (kromě simulačního modelu „Rosenbrock“) převládá hodnota parametru „GenerovatDleRozptylu = Nepravda“. Je tím tedy míněno volit generování hodnot u rozhodovacích proměnných nového prvku v celém příslušném intervalu rozhodovací proměnné. Stejné doporučení bylo prokázáno i u dodatečného testování sérií na simulačním modelu „Penalizace“ (viz Tabulka 13-20).

V následující tabulce (viz Tabulka 13-24) jsou uvedeny doporučené hodnoty parametru v závislosti na jednotlivých simulačních modelech. Tento přístup byl zejména vhodný pro těžký průběh „Ackleyho“ účelové funkce, kdy byl s větší pravděpodobností vygenerován prvek mimo údolí lokálního extrému.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Generovat dle rozptylu	Nepravda	Nepravda	Pravda	Nepravda	Nepravda	Nepravda	Nepravda

Tabulka 13-24 Doporučené hodnoty u parametru „Generovat dle rozptylu“ - Simulated Annealing

13.5.1.6 Snižovat teplotu jen při přijetí horšího řešení

V optimalizačním algoritmu simulovaného žíhání je implementována možnost snižovat teplotu (snížení pravděpodobnosti přijetí horšího řešení) jen tehdy, pokud je náhodně vygenerované číslo (podle rovnoměrného rozdělení) menší než specifikovaná pravděpodobnost přijetí horšího řešení nebo zda bude teplota snižována vždy, pokud dojde k vygenerování prvku s horší hodnotou účelové funkce než je hodnota účelové funkce předchozího prvku. Z grafu úspěšnosti sérií v závislosti na hodnotě parametru snižování teploty jen při přijetí horšího řešení (viz **příloha** – kapitola 6.5.6 Snižovat teplotu jen při přijetí horšího řešení, Obr. 6-18 Úspěšnost sérií s hodnotou parametru „Snižovat teplotu jen při přijetí horšího řešení“ - Simulated Annealing) lze vyvodit, že úspěšnost optimalizace není významně závislá na volbě jednotlivé varianty tohoto parametru.

V tabulce doporučených hodnot parametru **SnizovatTeplotuJenPriPrijetiHorsihoReseni** (viz Tabulka 13-24) jsou pro úplnost uvedeny hodnoty, které byly získány z provedených sérií optimalizačního algoritmu na vybraných simulačních modelech.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Snižovat teplotu jen při přijetí horšího řešení	Pravda	Pravda	Nepravda	Pravda	Nepravda	Nepravda	Nepravda

Tabulka 13-25 Doporučené hodnoty u parametru „Snižovat teplotu jen při přijetí horšího řešení“ - Simulated Annealing

13.5.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

Na základě následující tabulky a předchozích poznatků z analýz lze konstatovat, že největší vliv na úspěšnost algoritmu simulovaného žíhání má parametr „MinimalniTeplota“. V nejlepších sériích dominuje nejnižší hodnota tohoto parametru. Druhým základním parametrem ovlivňujícím úspěšnost je parametr určující, zda nový prvek bude generován podle rozptylu nebo jestli hodnoty rozhodovacích proměnných u nového prvku budou generovány v celém příslušném intervalu rozhodovací proměnné.

Název modelu	Rozptyl	Měnit jen jeden parametr	Beta	Minimální teplota	Generovat dle rozptylu	Snižovat teplotu jen při přijetí horšího řešení
DeJong	7	NEPRAVDA	0.52	0.01	NEPRAVDA	NEPRAVDA
Michalewicz	3	PRAVDA	0.52	0.01	NEPRAVDA	NEPRAVDA
Rosenbrock	7	PRAVDA	0.52	0.01	PRAVDA	PRAVDA
Ackley	13	PRAVDA	0.8	0.01	NEPRAVDA	NEPRAVDA
Doprava	5	PRAVDA	0.38	0.01	NEPRAVDA	NEPRAVDA
Penalizace	1	PRAVDA	0.94	0.01	NEPRAVDA	NEPRAVDA
VyrobniLinka	1	NEPRAVDA	0.8	0.25	NEPRAVDA	NEPRAVDA

Tabulka 13-26 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Simulated Annealing

13.5.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Z pěti vybraných vhodných nastavení optimalizačního algoritmu (viz Tabulka 13-27) umístěných na nejvyšším žebříčku hodnot kvality posuzované podle souhrnné funkce ƒ lze na základě analýz provedených u testovaných sérií a následující tabulky jako univerzální nastavení doporučit, negenerovat nové prvky podle rozptylu a používat nízkou hodnotu minimální teploty – v našem případě se nejvíce osvědčila hodnota 0.01.

Rozptyl	Měnit jen jeden parametr	Beta	Minimální teplota	Generovat dle rozptylu	Snižovat teplotu jen při přijetí horšího řešení	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
1	NEPRAVDA	0.94	0.01	NEPRAVDA	NEPRAVDA	0.278647153
1	PRAVDA	0.94	0.01	NEPRAVDA	NEPRAVDA	0.294809446
13	NEPRAVDA	0.52	0.01	NEPRAVDA	PRAVDA	0.321242266
9	NEPRAVDA	0.66	0.01	NEPRAVDA	NEPRAVDA	0.343691988
5	NEPRAVDA	0.38	0.01	NEPRAVDA	NEPRAVDA	0.351940865

Tabulka 13-27 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Simulated Annealing

13.6 Local Search

Algoritmus Local Search je další modifikací pseudogradientních metod. Podstatou je generování prvků v okolí dosud nejlepšího nalezeného prvku (náhodně s parametrem rozptyl) ve všech iteracích, na rozdíl od algoritmu Hill Climbing a Tabu Search, kde byl generován nový prvek v okolí předchozího prvku. Optimalizační algoritmus Local Search dosahoval přibližně stejné úspěšnosti jako oba předchozí zmiňované optimalizační algoritmy.

13.6.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Fakt, že algoritmus předčasně konverguje do lokálního extrému u těžších průběhů účelové funkce, potvrzuje i graf absolutní úspěšnosti optimalizačních algoritmů testovaných na všech simulačních modelech (viz Obr. 11-4 Absolutní úspěšnost sérií algoritmů na jednotlivých modelech na základě nalezení optima). Graf absolutní úspěšnosti byl zvolen z důvodu omezení vlivu špatného nastavení parametrů optimalizačního algoritmu. Výsledky ukazují, že u těžších průběhů účelové funkce („Michalewicz, Ackley, Doprava, Penalizace“) má optimalizační algoritmus sklon k předčasné konvergenci. Pro doplnění uvádím, že optimalizační algoritmus Local Search u simulačních modelů „Penalizace“ nedosáhl ani jedné absolutně úspěšné série z testovaných sérií, a proto ve sloupcovém grafu nejsou vykresleny žádné sloupce identifikující úspěšnost algoritmu u tohoto simulačního modelu. U lehkých průběhů účelové funkce je tento algoritmus účinný.

13.6.1.1 Rozptyl

Za účelem zachování stejných podmínek pro všechny optimalizační algoritmy byly provedeny pouze dvě série pro každou hodnotu rozptylu, protože počet provedených sérií je závislý na počtu parametrů. Algoritmus obsahuje pouze dva parametry – **Rozptyl** a **První populace náhodně**. Proto mohla hodnota úspěšnosti nabývat pouze 0 %, 50 % nebo 100 %. Ze sloupcového grafu vyplývá, že v případě lehkých průběhů účelové funkce pro simulační model „De Jong“ lze doporučit prakticky kteroukoliv nižší hodnotu rozptylu menší než 12 včetně (viz **příloha** – kapitola 6.6.1 Rozptyl, Obr. 6-19 Úspěšnost sérií s hodnotou parametru „Rozptyl“ – Local Search). U simulačního modelu „Rosenbrock“ bych doporučil všechny hodnoty rozptylu od 1÷11. U případu simulačního modelu „Ackley“ byl předpoklad, že bude zapotřebí většího rozptylu, aby prvek mohl uniknout z údolí lokálního minima a dostat se do údolí globálního minima. Protože povrch účelové funkce u simulačního modelu „VyrobniLinka“ není příliš členitý, dalo se očekávat, že nejlépe zafungují série s nižší hodnotou rozptylu. U jediného simulačního modelu „Penalizace“ se nevyskytla ani jedna absolutně úspěšná série.

V následující tabulce (viz Tabulka 13-28) jsou uvedeny doporučené hodnoty parametru „Rozptyl“. Pokud jsou u simulačních modelů doporučené hodnoty uvedeny v závorce „Penalizace“ jsou tyto hodnota získány ze sérií, které nebyly absolutně úspěšné, ale byly nejlepší vůči ostatním provedeným sériím na konkrétním simulačním modelu.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Rozptyl	[1,12] Krok 1	[3,8] Krok 1	[1,11] Krok 1	[10,11] ∪ [15,17] ∪ [21,22] Krok 1	[4,7] Krok 1	$*\{14\}$	[1,4] Krok 1

Tabulka 13-28 Doporučené hodnoty u parametru „Rozptyl“ - Local Search

V případě simulačního modelu „Penalizace“ byly dodatečné testovány hodnoty rozptylu 40÷100 s krokem 10. Z výsledků experimentování vyplynulo, že kvalita sérií se postupně zhoršovala s narůstající hodnotou rozptylu.

13.6.1 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

V následující tabulce (viz Tabulka 13-29) jsou uvedena vhodná nastavení parametru „Rozptyl“ pro jednotlivé simulační modely.

Název modelu	Rozptyl
DeJong	5
Michalewicz	4
Rosenbrock	2
Ackley	10
Doprava	4
Penalizace	14
VyrobniLinka	5

Tabulka 13-29 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Local Search

13.6.2 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Z následující tabulky (viz Tabulka 13-30) 5 vhodných nastavení parametru „Rozptyl“ pro všechny vybrané simulační modely u algoritmu Local Search vyplývá, že v případě testovaných simulačních modelů je lepší preferovat menší hodnoty rozptylu.

Rozptyl	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
6	0.73714395
3	0.745137839
4	0.759233609
5	0.773888607
10	0.847258767

Tabulka 13-30 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Local Search

13.7 Evolution Strategy

V případě Evoluční strategie se nový jedinec (**potomek**) generuje jako mutace původního jedince (**rodiče**) tak, že k jednotlivým původním složkám jedince jsou přičteny náhodné hodnoty generované na základě normálního rozdělení. U mutace jedince bylo implementováno Rechenbergovo adaptivní pravidlo jedné pětiny úspěšnosti upravující hodnoty směrodatných odchylek, zvlášť pro každou dimenzi, na základě vypočtené relativní četnosti úspěchů. V případě, že vypočtená relativní četnost úspěchů je menší než specifikovaná konstanta, bude uskutečněno jemnější prohledávání prostoru - **intenzifikace** (snaha zlepšovat dosavadní známé řešení pomocí malých změn, které vedou k novým podobným zlepšeným prvkům). Pokud vypočtená relativní četnost úspěchů

byla větší než definovaná konstanta, dochází k širšímu prohledávání prostoru tzv. **diverzifikaci**. Pokud vypočtená relativní četnost úspěchů byla shodná s definovanou konstantou, směrodatné odchylky se nemění.

Algoritmus je implementován v podobě varianty postupné vývojové strategie, kdy je výsledná populace tvořena určitým počtem jedinců s nejlepšími funkčními hodnotami ze všech jedinců jak rodičovské populace, tak populace potomků. V literatuře [14] se uvádí, že varianta postupné vývojové strategie (koeexistence rodičů a potomků) konverguje rychleji než varianta generační vývojové strategie (náhrada celé populace rodičů novými potomky). Nevýhodou je, že má větší tendenci ukončit prohledávání v lokálním extrému.

V literatuře (viz [14]) se doporučuje, aby velikost populace potomků byla volena několikanásobně větší než velikost rodičovské populace. V algoritmu je použita selekce tzv. turnajovým výběrem - výběr lepšího jedince z dvojice.

13.7.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

U algoritmu na základě hodnot úspěšnosti jednotlivých parametrů nelze podle grafů jednoznačně identifikovat, který parametr má zásadní vliv na chování algoritmu. Jednotlivé parametry totiž v grafech dosahují skoro stejné hodnoty úspěšnosti při hledání globálního optima.

13.7.1.1 Velikost populace

V optimalizačním algoritmu Evoluční strategie byla použita varianta **postupné vývojové strategie**, kdy je **výsledná** populace tvořena určitými jedinci, kteří jsou vybráni pomocí **turnajové selekce** z populace vzniklé spojením populace rodičů a populace potomků. Parametr **VelikostPopulace** určuje, kolik vítězných jedinců bude vybráno pomocí **turnajové selekce** do **výsledné** populace.

V případě implementovaného algoritmu Evoluční strategie vede větší hodnota parametru VelikostPopulace k menší pravděpodobnosti výběru nejlepšího jedince ze sloučených populací rodičů a potomků. Tím se částečně zabraňuje **elitářství** (výběru nejlepších jedinců z populace do následujících populací) a je podpořena **diverzita** řešení (různorodost navrhovaných řešení). Na základě těchto tvrzení, lze konstatovat, že v případě těžších průběhů účelové funkce by měly být voleny vyšší hodnoty tohoto parametru. Toto doporučení se potvrdilo na výsledcích provedených sérií u simulačních modelů „Ackley“ a „Michalewicz“, kde v úspěšných sériích převládají vyšší hodnoty tohoto parametru.

U simulačního modelu „Ackley“ byla komplikovanost v multimodálnosti účelové funkce, kde větší hodnota parametru velikosti populace umožnila odskočení i do vzdálenějších údolí s globálním minimem. U simulačního modelu „Michalewicz“ se nacházely rovinné oblasti a malé hodnoty parametru VelikostPopulace by mohly zapříčinit preferování nejlepších jedinců, což by vedlo k tomu, že algoritmus by předčasně konvergoval a nedostal by se do vzdálenějších oblastí koryt, kde se nachází globální minimum. Z provedených experimentů také vyplývá, že je vhodné u simulačního modelu „VyrobniLinka“ volit vyšší hodnoty parametru velikosti populace. Zde byl původní předpoklad volby menší hodnoty vzhledem k méně náročnému průběhu účelové funkce.

Z grafu úspěšnosti sérií (viz **příloha** – kapitola 6.7.1 Velikost populace, Obr. 6-20 Úspěšnost sérií s hodnotou parametru „Velikost populace“ – Evolution Strategy) s hodnotou parametru **VelikostPopulace** a minimálního rozdílu hodnot úspěšnosti jednotlivých sloupců vyplývá, že výše hodnoty parametru, určujícího velikost výsledné populace, nemá zásadní vliv na úspěšnost algoritmu. Pouze u jediného simulačního modelu „Penalizace“ nebyla nalezena ani jedna absolutně úspěšná série. Hodnota úspěšnosti nejlepší nalezené série u simulačního „Penalizace“ byla 80 %.

V následující tabulce (viz Tabulka 13-31) jsou uvedeny doporučené hodnoty parametru Velikost populace vycházející z předchozího grafu. V závorce u simulačního modelu „Penalizace“ je uvedena doporučená hodnota

určující velikost populace z provedených sérií, které nebyly absolutně úspěšné, ale byly nejlepší vůči ostatním provedeným sériím na simulačním modelu.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Velikost populace	[1,21] Krok 1	[1,21] Krok 1	[1,21] Krok 1	[1,21] Krok 1	[1,19] Krok 1	$*\{11,15,21\}$	[5,21] Krok 1

Tabulka 13-31 Doporučené hodnoty u parametru „Velikost populace“ – Evolution Strategy

13.7.1.2 Mmp – počet potomků

Hodnota parametru **Mmp** vyjadřuje, kolik bude vytvořeno potomků (prvků) na základě rodičovské populace.

V literatuře se doporučuje, aby velikost populace potomků byla volena několikanásobně větší než velikost rodičovské populace. [14]

Na rozdíl od parametru **VelikostPopulace**, výše hodnoty parametru **Mmp** u simulačních modelů „Michalewicz“, „Ackley“ a „Doprava“ výrazněji ovlivňuje chování algoritmu. Z grafu úspěšnosti (**viz příloha** – kapitola 6.7.2 Mmp – počet potomků, Obr. 6-21 Úspěšnost sérií s hodnotou parametru „Mmp“ - Evolution Strategy) vyplývá, že největší úspěšnosti u simulačních modelů „Ackley“ a „Doprava“ a částečně i „Penalizace“ dosáhly série s počtem jedinců k páření „Mmp=1“.

Znamená to tedy, že u těžších průběhů účelové funkce byl v sériích preferován výběr jednoho jedince, který byl vybrán ze zápasu pomocí turnajové selekce. Volbou „Mmp=1“ se můžeme vystavovat vyšší pravděpodobnosti předčasné konvergence díky intenzifikaci – zlepšování dosavadní populace pomocí malých přírůstků ve formě preference kvalitního jedince. Propojením nastavení malé hodnoty parametru **k** (počet jedinců v turnaji, kdy hodnota parametru „k=1“ turnajovou selekci degeneruje na náhodný výběr jedinců – podpora diverzity) a malé hodnoty **Mmp** (intenzifikace pomocí „Mmp=1“) může být vyváženým kompromisem mezi a podporou diverzity a intenzifikace na straně druhé u těžších průběhů účelové funkce.

V následující tabulce (viz Tabulka 13-32) jsou uvedeny nalezené vhodné hodnoty parametru Mmp algoritmu Evoluční strategie vyjadřujícího počet jedinců k páření u jednotlivých sérií provedených na všech simulačních modelech.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Mmp	[1,21] Krok 1	[1,21] Krok 1	[1,21] Krok 1	[1,21] Krok 1	[1,21] Krok 1	$*\{1,5,7,19\}$	[1,21] Krok 1

Tabulka 13-32 Doporučené hodnoty u parametru „Mmp“ - Evolution Strategy

13.7.1.3 q – počet předcházejících úspěchů

U mutace rodiče (pomocí normálního rozdělení), pomocí níž je vygenerován nový potomek, bylo implementováno Rechenbergovo adaptivní pravidlo jedné pětiny úspěšnosti upravující hodnoty směrodatných odchylek zvlášť pro každou dimenzi na základě vypočtené relativní četnosti úspěchů. Relativní četnost úspěchů je spočtena ze seznamu, v němž jsou ukládány informace o úspěších při generování lepšího potomka než rodiče. Úspěchem se míní vygenerování lepšího potomka, než byl jeho rodič. V případě úspěchu je do vektoru umístěna 1. Pokud tomu tak není, potomek je horší než jeho rodič, je do seznamu vložena 0. Relativní četnost úspěchů je spočtena tak, že všechny hodnoty v seznamu jsou sečteny a vyděleny délkou seznamu tj. počet složek v seznamu. Parametr **q** určuje velikost seznamu úspěchů, tj. kolik má být sledováno předchozích úspěchů v populaci. Pokud vypočtená relativní četnost úspěchů je menší, než specifikovaná, bude uskutečněno jemnější prohledávání prostoru - intenzifikace. Pokud vypočtená relativní četnost úspěchů byla větší než definovaná konstanta, dochází k širšímu prohledávání prostoru - diverzifikace. Pokud vypočtená relativní četnost úspěchů byla shodná s definovanou konstantou, směrodatné odchylky se nemění.

Z následujícího grafu (viz **příloha** – kapitola 6.7.3 q – počet předcházejících úspěchů, Obr. 6-22 Úspěšnost sérií s hodnotou parametru „q“ - Evolution Strategy) vyplývá, že výše hodnoty parametru q nemá zásadní vliv na úspěšnost algoritmu Evoluční strategie u lehčích průběhů účelové funkce. V případě multimodální „Ackleyho“ funkce a simulačního modelu „Doprava“ je patrné, že s rostoucí hodnotou parametru se zvyšovala úspěšnost provedených sérií. V následující tabulce (viz Tabulka 13-33) jsou zobrazeny nejlepší nalezené hodnoty parametru „q“ u provedených sérií u jednotlivých simulačních modelů.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
q	[1,21] Krok 5	[1,21] Krok 5	[1,21] Krok 5	[1,21] Krok 5	[1,21] Krok 5	∗ {21}	[1,21] Krok 5

Tabulka 13-33 Doporučené hodnoty u parametru „q“ - Evolution Strategy

13.7.1.4 k – počet jedinců v turnaji

K selekci jedinců z populace, která vznikla smísením rodičovské populace a populace potomků, byla použita turnajová selekce se substitucí (v populaci se může vyskytnout i kopie jedince). Parametr **k** určuje s kolika soupeři, kteří byli vybráni náhodně, bude vybraný jedinec bojovat. Do výběru se pak dostává jedinec, který zvítězil nade všemi ostatními soupeři. Jedná se tedy o preferenci silnějších jedinců. Z grafu úspěšnosti sérií s hodnotami parametru **k** vidíme, že u všech simulačních modelů, kromě jediného - „Ackley“, byly série s „k=1“ silně neúspěšné, vzhledem k ostatním hodnotám parametru.

Je zvláštní, že tento parametr s hodnotou „k=1“ dosahoval u absolutně úspěšných sérií (viz **příloha** – kapitola 6.7.4 k – počet jedinců v turnaji, Obr. 6-23 Úspěšnost sérií s hodnotou parametru „k“ - Evolution Strategy) u simulačních modelů vytvořených podle funkce („DeJong, Michalewicz, Rosenbrock, Ackley“) skoro stejné hodnoty úspěšnosti. Z výše hodnot úspěšnosti u jednotlivých datových řad parametru **k** u funkcí („DeJong, Michalewicz, Rosenbrock“), lze usuzovat, že výše hodnoty parametru, kromě hodnoty 1, nemá zásadní vliv na úspěšnost sérií. Jediný simulační model „Ackleyho“ funkce upřednostňoval u provedených sérií hodnotu parametru „k=1“.

Jestliže „k=1“, turnajová selekce degeneruje na náhodný výběr jedinců a každý kandidát řešení se průměrně dostane do populace k páření jedenkrát. Se vzrůstající hodnotou **k** selekční tlak vzrůstá: jedinci s dobrou hodnotou fitness tvoří větší a větší skupinu potomků, zatímco šance na reprodukci horších kandidátů na řešení klesá. [15]

Následující tabulka (viz Tabulka 13-34) obsahuje nejlepší hodnoty koeficientu k u absolutně úspěšných sérií umístěných v předchozím sloupcovém grafu (viz **příloha** – Obr. 6-23 Úspěšnost sérií s hodnotou parametru „k“ - Evolution Strategy). Tabulka je doplněna o nejlepší hodnotu koeficientu **k** (v závorce s hvězdičkou) z nejlepších nalezených sérií (výběr pouze sérií, které dosahovaly nejvyšší úspěšnosti při hledání globálního optima) na simulačním modelu „Penalizace“.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
k	[5,21] Krok 4	[5,21] Krok 4	[5,21] Krok 4	[1,21] Krok 4	[5,21] Krok 4	∗ {5}	[5,21] Krok 4

Tabulka 13-34 Doporučené hodnoty u parametru „k“ - Evolution Strategy

13.7.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

Pokud srovnáme hodnoty jednotlivých parametrů nejlepších sérií umístěných v následující tabulce (viz Tabulka 13-35) zjistíme, že hodnoty jsou skoro ve většině případů odlišné. Mohli bychom tedy vyvodit závěr, že algoritmus Evoluční strategie není ve velké míře závislý na hodnotách svých jednotlivých parametrů.

Jediným doporučením může být nenastavovat „q=1“ u všech simulačních modelů kromě simulačního modelu „Ackley“, kde série s touto hodnotou dosahovaly nejvyšší úspěšnosti ze všech testovaných nastavení u tohoto simulačního modelu.

Název modelu	Velikost populace	Mmp	q	k
DeJong	3	1	1	9
Michalewicz	5	5	1	13
Rosenbrock	3	1	1	21
Ackley	7	17	21	17
Doprava	3	7	1	17
Penalizace	21	5	21	5
VyrobniLinka	19	9	11	9

Tabulka 13-35 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Evolution Strategy

Algoritmus Evoluční strategie lze na základě testování všech optimalizačních algoritmů doporučit, jako efektivní a použitelný algoritmus na všechny testované typy průběhů účelových funkcí. Lze ho také doporučit pro jeho nenáročnost na nastavení jednotlivých parametrů algoritmu. Nespornou výhodou algoritmu je jeho schopnost uniknout z lokálních extrémů účelové funkce.

Poznámka: Jako druhou variantu optimalizačního neevolučního algoritmu bych doporučil algoritmus simulovaného žíhání.

13.7.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

Ve výběru pěti vhodných nalezených sérií provedených u všech simulačních modelů (viz Tabulka 13-36) se do předního žebříčku dostaly série s hodnotou parametru „q=1“. Podle hodnot v předchozím grafu úspěšnosti sérií s hodnotou parametru **q** (viz **příloha** – kapitola 6.7.3 q – počet předcházejících úspěchů, Obr. 6-22 Úspěšnost sérií s hodnotou parametru „q“ - Evolution Strategy) ale vyplývá, že výše hodnoty parametru **q** nemá velký vliv na úspěšnost při hledání optima. U algoritmu na základě hodnot úspěšnosti jednotlivých parametrů nelze jednoznačně definovat, který parametr má na chování algoritmu největší vliv. Jednotlivé parametry totiž dosahují skoro stejné hodnoty úspěšnosti při hledání globálního optima.

Velikost populace	Mmp	q	k	Součet hodnot souhrnné funkce ƒ u všech simulačních modelů
5	5	1	17	0.409612055
5	15	1	5	0.410918701
9	5	1	17	0.413349094
9	1	1	9	0.413997462
7	3	1	9	0.41748548

Tabulka 13-36 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Evolution Strategy

13.8 Differential Evolution

Algoritmus využívá tradiční evoluční operátory křížení, mutace a selekce. V algoritmu byl využit princip adaptivity mutačního parametru. **Nový** jedinec (v případě evolučních algoritmů se na místo termínu prvek užívá termín jedinec) vzniká **mutací** nejlepšího jedince se čtyřmi náhodně vybranými jedinci, kteří jsou různí od nejlepšího prvku v populaci a aktuálního vybraného prvku z populace – princip metody **BEST**. Tento princip má v počátečním stádiu uchovávat diverzitu prohledávání (**diverzifikace** - explorace) a v pozdějším stádiu zvyšuje intenzitu prohledávání (**intenzifikace** - exploatace).

Princip diferenciální evoluce je založen na **selekci** (výběru) lepšího jedince ze dvou jedinců - **rodiče** a jeho **potomka**. Potomek vznikl křížením **rodiče** a **nového** vytvořeného jedince, který vznikl pomocí mutace jedinců – metoda BEST. Lepší jedinec z těchto dvou porovnávaných jedinců je následně zařazen do populace, která kompletně nahradí populaci rodičů.

13.8.1 Nalezení vhodného nastavení parametru optimalizačního algoritmu pro každý simulační model

Experimentální výsledky i zkušenosti ukazují, že diferenciální evoluce konverguje rychleji než jiné stochastické algoritmy (viz literatura [14]). S tímto tvrzením lze souhlasit. Pokud ale porovnáme výsledky grafu průměrné úspěšnosti algoritmů při nalezení optima (viz Obr. 11-2), průměru hodnot čtvrtého kritéria rychlosti nalezení optima (Obr. 11-14) a průměru hodnot z ohodnocení na základě třetího kritéria u absolutně úspěšných sérií (viz Obr. 11-13), lze konstatovat, že testovaná forma algoritmu Diferenciální evoluce je rychlá v nalezení optima (kromě algoritmu Downhill Simplex), ale u jednodušších průběhů účelové funkce poskytuje horší výsledky než ostatní optimalizační algoritmy. U těžšího simulačního modelu „Penalizace“ algoritmus zafungoval o mnoho lépe než ostatní algoritmy založené na pseudogradientním přístupu (výběr všech sérií, kromě absolutně úspěšných). Hodnota f_3 poskytující přehled o kvalitě výsledků poskytnutých algoritmem, byla srovnatelná s velmi efektivním dále hodnoceným algoritmem Evoluční strategie.

13.8.1.1 Velikost populace

V algoritmu je uplatněn princip vytváření nové populace jedinců na základě zvolení lepšího jedince z dvou konkurentů – rodiče a potomka. Parametr **VelikostPopulace** určuje, kolik bude do výsledné populace umístěno přeživších jedinců na základě selekce rodič vs. potomek. Z grafu úspěšnosti (viz **příloha** – kapitola 6.8.1 Velikost populace, Obr. 6-24 Úspěšnost sérií s hodnotou parametru „VelikostPopulace“ – Differential Evolution) je zřejmé, že nejlépe dopadly série, kde byla zvolena největší populace jedinců. Totálním neúspěchem dopadla volba populace o velikosti 1.

Pro úplnost doplním, že hodnoty úspěšnosti nejlepší nalezené série u jednotlivých simulačních modelů jsou: „Ackley“: úspěšnost 80 %, „Penalizace“: úspěšnost 20 %.

Následující tabulka (viz Tabulka 13-37) obsahuje nalezené vhodné hodnoty parametru VelikostPopulace z testovaných sérií. Je zřejmé, že nejvíce se osvědčilo u vybraných simulačních modelů volit nastavení parametru algoritmu s větší hodnotou velikosti populace.

Sami tvůrci algoritmu Storn a Price doporučují volit hodnoty velikosti populace pětinásobku až desetinásobku počtu rozhodovacích proměnných (minimálně však 4, aby bylo zajištěno, že algoritmus Diferenciální evoluce bude mít dostatek odlišných prvků, se kterými může pracovat (viz literatura [33]).

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Velikost populace	[24,30] Krok 2	[22,30] Krok 2	[18,28] ∪ {30} Krok 2	∗ {24} ∪ ∗ {28}	[24,30] Krok 2	∗ [20,30] Krok 2	[18,30] Krok 2

Tabulka 13-37 Doporučené hodnoty u parametru „VelikostPopulace“ – Differential Evolution

13.8.1.2 Koef_F – koeficient adaptivního pravidla

Parametr označovaný jako **Koef_F** je koeficient využívaný při mutaci původního jedince. Koeficient je měněn na základě adaptivního pravidla (podle Ali a Törna), podle kterého je hodnota parametru měněna v každém iteračním kroku na základě minimální a maximální hodnoty účelové funkce v populaci. Díky tomuto principu je v počáteční fázi algoritmu uchována **diverzita** prohledávání a v pozdějším stádiu se zvyšuje **intenzita** prohledávání.

Na základě výsledků následujícího grafu (viz **příloha** – kapitola 6.8.2 Koef_F – koeficient adaptivního pravidla, Obr. 6-25 Úspěšnost sérií s hodnotou parametru „Koef_F“ - Differential Evolution) lze v případě testovaných simulačních modelů doporučit hodnotu koeficientu mutace „Koef_F = 0.55“.

Sami tvůrci [33] doporučují volit „Koef_F = 0.5“, což odpovídá dosaženým výsledkům provedených analýz. Pokud algoritmus předčasně konverguje, je doporučeno (viz literatura [33]) zvyšovat parametr a/nebo velikost populace. Pokud $0.4 > \text{Koef\_F}$ je prohledávání jen ojediněle efektivní. [33]

Sami tvůrci algoritmu užívají pro odlišné simulační modely ve formě testovacích funkcí různých hodnot parametrů ($0.5 \leq \text{Koef\_F} \leq 1$).

Souhrnně u všech simulačních modelů je preferována hodnota parametru „Koef_F = 0.55“ (viz **příloha** – kapitola 6.8.2 Koef_F – koeficient adaptivního pravidla, Obr. 6-25 Úspěšnost sérií s hodnotou parametru „Koef_F“ - Differential Evolution). Jedinou výjimkou je hodnota „Koef_F = 0.22“ u simulačního modelu „Penalizace“. V následující tabulce (viz Tabulka 13-38) jsou uvedeny doporučené hodnoty parametru Koef_F.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Koef_F	[0.33,0.66] Krok 0.11	[0.55,0.66] Krok 0.11	[0.55,0.66] Krok 0.11	$*\{0.55\}$	[0.55,0.66] Krok 0.11	$*\{0.22\}$	[0.33,0.66] Krok 0.11

Tabulka 13-38 Doporučené hodnoty u parametru „Koef_F“ - Differential Evolution

13.8.1.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných

Nový prvek - potomek vzniká křížením nově vygenerovaného (zmutovaného) jedince a původního rodiče. Potomkův genom (vektor složený z hodnot rozhodovacích proměnných) skládající se z jednotlivých genů (jednotlivé rozhodovací proměnné) vzniká tak, že pokud náhodně vygenerované číslo v intervalu $[0,1)$ je menší nebo rovno definované pravděpodobnosti, tj., parametr **Koef_C** nebo pokud došlo k vygenerování stejného čísla, jako je index rozhodovací proměnné, potomek zdědí gen nově vygenerovaného zmutovaného jedince (dochází ke křížení genu). V opačném případě dědí gen po rodiči.

Autoři algoritmu uvádí, že první volbou by měla být pravděpodobnost nahrazení souřadnic rozhodovacích proměnných rovna hodnotě „Koef_C = 0.1“, ale zároveň uvádí, že vyšší hodnota parametru Koef_C často vede k rychlejší konvergenci. Doporučují tedy dále otestovat nastavení „Koef_C = 0.9“ nebo „Koef_C = 1“ a zjistit, které nastavení je rychlejší. [33]

Pokud je tedy nastavena nižší pravděpodobnost nahrazení souřadnic rozhodovacích proměnných, potomek bude s menší pravděpodobností dědit geny od nově zmutovaného jedince, který vznikl **mutací** nejlepšího jedince a dalších čtyř navzájem různých náhodně zvolených jedinců. To znamená, že bude spíše dědit geny od svého rodiče, což povede k tomu, že algoritmus nebude udržovat diverzitu populace.

Na základě výsledků grafu úspěšnosti (viz **příloha** – kapitola 6.8.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných, Obr. 6-26 Úspěšnost sérií s hodnotou parametru „Koef_C“ - Differential Evolution) lze tvrdit, že v případě simulačních modelů „De Jong, Michalewicz, Rosenbrock, Doprava“ se osvědčuje nastavení parametru na vyšší hodnotu „Koef_C = 0.77“ (větší diverzita v populaci). Pro případ simulačního modelu „Ackley“ (multimodální průběh účelové funkce) jsou v následující tabulce (viz Tabulka 13-39) uvedeny různorodé hodnoty, které dosahovaly shodné 80 % úspěšnosti při nalezení optima ($f_1 = 0.2$). V převaze jsou opět větší hodnoty parametru Koef_C, což vede k domněnce, že je v našem případě vybraných simulačních modelů lepší používat větší hodnoty pravděpodobnosti nahrazení souřadnic rozhodovacích proměnných u potomka.

Název modelu / Parametr	DeJong	Michalewicz	Rosenbrock	Ackley	Doprava	Penalizace	VyrobniLinka
Kontrakce	[0.33,0.88] Krok 0.11	[0.44,0.88] Krok 0.11	[0.44,0.88] Krok 0.11	∗ [0.44,0.88] Krok 0.11	[0.44,0.77] Krok 0.11	∗ {0.11} ∪ ∗ [0.55,0.88] Krok 0.11	[0.22,0.99] Krok 0.11

Tabulka 13-39 Doporučené hodnotyu parametru „Koef_C“ - Differential Evolution

13.8.2 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely

U předchozí analýzy velikosti populace bylo zjištěno, že u algoritmu Diferenciální evoluce je lepší preferovat větší velikost populace. U všech simulačních modelů byla s převahou preferována hodnota parametru „Koef_F = 0.55“. Rozdíl doporučených hodnot sérií a nejlepších nalezených sérií nastává v případě hodnot parametru pravděpodobnosti nahrazení souřadnic rozhodovacích proměnných u potomka. Z výše hodnot úspěšnosti v předcházejícím grafu úspěšnosti sérií s hodnotou parametru Koef_C (viz **příloha** - kapitola 6.8.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných, Obr. 6-26 Úspěšnost sérií s hodnotou parametru „Koef_C“ - Differential Evolution) a následující tabulky (Tabulka 13-40) lze usoudit, že parametr nemá až tak velký vliv na efektivitu při hledání globálního optima.

Název modelu	Velikost populace	Koef_F	Koef_C
DeJong	26	0.55	0.11
Michalewicz	16	0.55	0.77
Rosenbrock	22	0.55	0.55
Ackley	28	0.55	0.88
Doprava	24	0.55	0.55
Penalizace	26	0.22	0.11
VyrobniLinka	24	0.33	0.55

Tabulka 13-40 Nalezená vhodná nastavení všech parametrů algoritmu pro konkrétní simulační modely - Differential Evolution

13.8.3 Nalezená vhodná nastavení všech parametrů algoritmu pro všechny vybrané simulační modely

U tabulky pěti vhodných nastavení všech parametrů algoritmu Diferenciální evoluce (viz Tabulka 13-41) se potvrzují doporučované hodnoty z předchozích analýz jednotlivých parametrů.

Velikost populace	Koef_F	Koef_C	Součet hodnot souhrnné funkce f u všech simulačních modelů
28	0.55	0.44	0.484476879
24	0.55	0.66	0.496113662
26	0.55	0.33	0.501815807
18	0.55	0.55	0.507606601
30	0.66	0.55	0.509600904

Tabulka 13-41 Výběr 5 vhodných nastavení všech parametrů algoritmu pro všechny vybrané simulační modely - Differential Evolution

14 Přínosy disertační práce

V disertační práci byly analyzovány dílčí oblasti simulační optimalizace s cílem modifikovat vhodné optimalizační algoritmy pro simulace diskrétních výrobních procesů a systémů, doporučit jejich volbu a nastavení jejich parametrů pro různé účelové funkce. Jedním z důvodů vytvoření této disertační práce byl fakt, že se většina simulačních optimalizátorů chová jako „černá skříňka - Black-Box“, a nelze s jistotou určit, které optimalizační algoritmy byly při optimalizačním procesu použity, jaké efektivity při hledání optima dosahovaly, jaké bylo provedeno nastavení parametrů optimalizačního algoritmu atd. Je tedy možné, že v těchto simulačních optimalizátorech nemusel být vybrán efektivní optimalizační algoritmus a špatný výběr mohl vést k neefektivnímu prohledávání, k předčasné konvergenci do lokálního extrému namísto globálního optima, atd. V disertační práci také byly řešeny další zjištěné nedostatky v této oblasti. Výčet řešených nedostatků je uveden v kapitole 5 Teoretická východiska z hodnocení současného stavu.

Bylo testováno chování optimalizačních algoritmů na různých průbězích účelové funkce, a to za účelem doporučení vhodných optimalizačních algoritmů pro určité průběhy účelové funkce. Toto testování bylo prováděno v tzv. sériích (opakování optimalizačních experimentů, aby byl omezen vliv náhody implementovaný v optimalizačních algoritmech) pomocí vytvořené aplikace simulační optimalizace. Do této aplikace bylo implementováno grafické vyhodnocování výsledků sérií optimalizačních experimentů ve formě krabicových grafů a jejich vypočtených charakteristik pomocí tří hledisek – nalezená optima, rychlost nalezení optima, konvergence. Toto vizuální hodnocení nabízí rychlý přehled o kvalitě jednotlivých sérií.

Protože byl proveden velký počet sérií a implementované vizuální hodnocení pomocí krabicových grafů bylo značně neefektivní (někdy až nemožné), byla proto navržena metodika pro vyhodnocení sérií podle různých kritérií. Kritéria byla specifikována na základě analýzy chování optimalizačních algoritmů. Převážná většina vyhodnocení je dnes běžně prováděna pomocí standardní odchylky hodnoty účelové funkce nalezeného optima od hodnoty účelové funkce skutečného globálního optima. Není však brán zřetel na to, že praxe namísto matematické přesnosti, může být spokojena i s přijatelným řešením, které bude blízko globálního, namísto přesného určení globálního optima nebo bude záležet na rychlosti nalezení optima, atd.

Pro účely hodnocení byla také vytvořena aplikace pro zobrazení trojrozměrného grafu účelové funkce a zobrazení optimalizačního experimentu pomocí zvoleného optimalizačního algoritmu na testované účelové funkci. Na základě navržené metodiky pro provádění analýz a hodnocení výsledků sérií, byla formulována doporučení vztahující se k optimalizačním algoritmům.

14.1 Teoretické přínosy práce

Hlavní teoretické přínosy této práce vycházejí z důkladné analýzy oblasti simulační optimalizace, která zahrnuje:

- Identifikace základních atributů simulační optimalizace.
- Transformace těchto základních prvků, používaných převážně v oblasti simulační optimalizace u funkcí, takovým způsobem, aby mohly být aplikovány i na oblast strojírenské výroby využívající diskrétní simulační modely.
- Specifikace základních metod využívaných pro práci s omezujícími podmínkami a jejich transformace do podoby algoritmu.
- Specifikace základních problémů globální optimalizace, které mohou nastat při simulačním experimentování.
- Provedení analýzy u vybraných optimalizačních metod. Algoritmus Evoluční strategie se jeví jako nejuniverzálnější ze zkoumaných optimalizačních algoritmů.
- Modifikace vybraných optimalizačních metod za účelem využití v diskrétní simulaci a jejich transformace do podoby algoritmu (algoritmy jsou popsány pomocí kódu v pseudopascalu včetně obecných popisů principů algoritmů).

- Implementace základních principů z oblasti evolučních algoritmů do některých pseudogradientních optimalizačních algoritmů (viz **příloha** – kapitola 4.1 Implementované optimalizační metody) – vytváření celých skupin (populací) prvků (jedinců) namísto jediného prvku pro omezení vlivu předčasné konvergence, implementace generování nových prvků pomocí mutace prvku, specifikace modifikace algoritmů za účelem využití množiny optim (funkce UpdateOptimalSet, ExtractOptimalSet, apod.).
- Specifikace a transformace různých typů algoritmů pro přiřazení fitness. Přiřazení fitness na základě (viz **příloha** – kapitola 3.1 Přiřazení fitness):
    - Komparační funkce – dvě verze.
    - Pořadí.
    - Turnajového zápasu.
- Specifikace a transformace různých typů algoritmů selekce (viz **příloha** – kapitola 3.2 Selekce):
    - Zkrácený výběr.
    - Náhodná selekce.
    - Selekce přímo úměrná fitness - Ruletová selekce.
    - Turnajová selekce.
    - Selekce na základě uspořádání
- Navržení obecné struktury **aplikace simulační optimalizace** využívající předchozí popsané prvky globální optimalizace. Návrh se týká dvou oblastí:
    - **Simulační optimalizátor**, který slouží pro vyhledání vhodného řešení u simulačního modelu.
    - Vlastní **experimentální základna určená pro testování** chování zkoumaných optimalizačních algoritmů pro různé typy účelových funkcí.
- Provedení hromadných sérií optimalizačních experimentů pomocí optimalizačních algoritmů s využitím experimentální základny.
- Navržení, implementace a validace **metodiky** vyhodnocení optimalizačních experimentů provedených na experimentální základně podle různých kritérií.
- Navržení, implementace a validace **metodiky** nalezení vhodného nastavení parametrů optimalizačních algoritmů na základě optimalizačních experimentů (opakování optimalizačních experimentů v rámci série) provedených na experimentální základně.
- Závěry a doporučení jakým způsobem nastavit parametry testovaných optimalizačních algoritmů v závislosti na různých průbězích účelové funkce.

14.2 Praktické přínosy práce

Jak již bylo dříve uvedeno, v průběhu několika předchozích let bylo na katedře Průmyslového inženýrství a managementu Fakulty strojní Západočeské univerzity v Plzni řešena řada simulačních úloh pro průmyslovou praxi. Bohužel u většiny případů těchto simulačních modelů, vytvořených v simulačním software ARENA, nebylo možné plně využít integrovaný simulační optimalizátor OPTQuest, který je součástí simulačního systému. Důvodem bylo především to, že simulační optimalizátor umožňuje nastavovat jen vybrané typy parametrů simulačního modelu. Toto omezení je však pro praktické úlohy často nepřípustné.

Na základě nedostatků, zjištěných pomocí rešerše, a provedené analýzy v oblasti simulační optimalizace, byla po fázi návrhu potřeb simulační optimalizace, vytvořena otevřená softwarová **aplikace simulační optimalizace**, která do značné míry eliminuje zjištěné nedostatky. Tato aplikace obsahuje simulační optimalizátor, který bude využíván pro praktické simulační optimalizační úlohy používající diskrétní simulační model vytvořený v prostředí ARENA. Dále aplikace obsahuje vlastní experimentální základnu určenou pro testování chování implementovaných optimalizačních algoritmů. Pro účely testování vybraných optimalizačních algoritmů byly vytvořeny simulační modely (jak praktické tak i teoretické), včetně specifikací různých typů účelové funkce.

Výsledky analýzy prokázaly, že vhodná volba optimalizačního algoritmu a nastavení parametrů optimalizačního algoritmu výrazně ovlivní průběh optimalizačního procesu (automatického řízení simulačních experimentů).

15 Doporučení pro další postup

Na základě analýzy získaných výsledků sérií optimalizačních experimentů vybraných optimalizačních algoritmů na vybraných simulačních modelech byly vyvozeny následující doporučení pro další postup:

- Podpora učení simulačního optimalizátoru pomocí neuronové sítě.
- Testování optimalizačních algoritmů na více praktických příkladech a také složitějších případech.
- Modifikace optimalizačních algoritmů zkoumaných v rámci disertační práce na základě získaných poznatků z testování.
- Implementace více druhů optimalizačních algoritmů do aplikace simulační optimalizace, např. SOMA – Samoorganizující se migrační algoritmus, genetické algoritmy, atd.
- Testování různých typů selekcí a analýza jejich vlivu na chování optimalizačního algoritmu při hledání globálního optima.
- Přepínání optimalizačních algoritmů v závislosti na úspěchu, např. algoritmy se soutěžícími heuristikami (pravděpodobnost výběru heuristiky je úměrná její dosavadní úspěšnosti).
- Implementace ovládání různých typů simulačních modelů vytvořených v různých simulačních softwarech, např. PlantSimulation, Witness, atd.
- Implementace metodiky vyhodnocení hromadných experimentů do softwarové aplikace simulační optimalizace.

16 Závěr

Z výsledků testování optimalizačních algoritmů na různých typech účelových funkcí vyplývá, že optimalizační algoritmus Evoluční strategie je vhodným optimalizačním algoritmem pro téměř všechny testované průběhy účelových funkcí. Tento optimalizační algoritmus také poskytoval, podle ukazatele f_2 (rozdílu nalezeného lokálního extrému od globálního optima účelové funkce) u testovaných typů účelových funkcí, v průměru nejlepší řešení. Alternativou k evolučním algoritmům by mohl být algoritmus Simulovaného žíhání. Tento algoritmus, spolu s optimalizačním algoritmem Random Search, se umístil na předním místě testovaných optimalizačních algoritmů. Simulované žíhání má možnost uniknout z lokálního extrému díky implementované podstatě postupného snižování teploty, tedy postupného snižování pravděpodobnosti přijetí horšího řešení.

Na první pohled se může zdát, že algoritmus Random Search je také úspěšným algoritmem. Ve skutečnosti je tomu ale tak, že ukazatel úspěšnosti u tohoto optimalizačního algoritmu byl v našem případě ovlivněn poměrně malou velikostí prohledávaného prostoru. Spolu s možností zkoumat celý prohledávaný prostor a generovat stále jiné nové prvky, by bylo nepravděpodobné, že algoritmus nenalezne globální optimum. S rostoucím prohledávaným prostorem se však pravděpodobnost úspěchu při hledání globálního optima a při zachování počtu provedených experimentů bude u tohoto optimalizačního algoritmu značně snižovat.

Algoritmy založené na pseudo-gradientním prohledávání - Hill Climbing, Local Search, Tabu Search dosahovaly při hledání globálního extrému u jednoduchých průběhů účelové funkce přibližně stejné úspěšnosti. Je to dáno stejnou podstatou algoritmů. Implementací přístupu generování celých populací, namísto jediného prvku, u optimalizačních algoritmů Hill Climbing a Tabu Search, lze předejít předčasné konvergenci, kterou trpí optimalizační algoritmy založené na pseudo-gradientním přístupu.

Špatných výsledků dosahovaly optimalizační algoritmy Downhill Simplex a Diferenciální evoluce. U optimalizačního algoritmu Downhill Simplex bylo příčinou neúspěchu odchýlení od vypočteného původního směru díky zaokrouhlování souřadnic směrem k nejbližšímu možnému bodu, což bylo prokázáno na základě dalšího testování. Při stanovení menšího kroku v prohledávaném prostoru absolutní úspěšnost sérií optimalizačního algoritmu Downhill Simplex dosahovala dokonce srovnatelné hodnoty s absolutní úspěšností favorizovaného optimalizačního algoritmu Evoluční strategie. Problém zaokrouhlování může být řešen modifikací algoritmu. Modifikaci bude třeba odvodit a ověřit pomocí aproximace hodnot účelové funkce v nejbližších bodech.

U Diferenciální evoluce bylo důvodem neúspěchu při hledání globálního optima potlačení diverzity v prohledávání, které bylo zapříčiněno kopírováním identických jedinců do dalších populací. Je také nutné podotknout, že oba optimalizační algoritmy velmi rychle konvergovaly. Diferenciální evoluce dokonce dosahovala u sérií, kde nebylo nalezeno optimum, v rámci kvality poskytovaných výsledků (rozptyl hodnot účelové funkce nalezených výsledků) lepších hodnot, než optimalizační algoritmy založené na pseudogradientním přístupu.

Z grafů výsledků průměrné úspěšnosti jednotlivých optimalizačních algoritmů u jednotlivých simulačních modelů se potvrzuje, že chování optimalizačních algoritmů je silně závislé na průběhu účelové funkce. Pokud tedy řešíme nějaký optimalizační problém, je vhodné nejprve nasbírat co nejvíce informací o průběhu účelové funkce (účelová funkce je spojitá, multimodální, přibližně lineární, atd.). S ohledem na tyto informace lze pak zvolit vhodný optimalizační algoritmus. V mnoha případech simulační optimalizace u simulačních modelů z praxe však tyto informace k dispozici nemáme. V takových případech, s ohledem na provedené experimentování s vybranými optimalizačními algoritmy, se jeví jako nejlepší vybrat optimalizační algoritmus Evoluční strategie (zejména díky malé necitlivosti na nastavení parametrů v případě vybraných simulačních modelů) nebo Simulovaného žíhání. Algoritmus Random Search lze doporučit zejména v případech, kdy prohledávaný prostor není moc veliký a nemáme absolutně žádnou představu o průběhu účelové funkce.

17 Použitá literatura

[1] HOŘEJŠÍ, P. a CANDROVÁ, K., "Výzkum metod optimálního řízení distribuovaných simulačních modelů výrobních procesů a jejich integrace do konceptu digitálního podniku - Experimenty s modelem virtuální dílny." *Interní grant.* Plzeň : Západočeská univerzita v Plzni/FST/KPV, 2006.
[2] , OR/MS TODAY. [Online] INFORMS - Institute for Operations Research and the Management Sciences, Říjen 2011. [Citace: 15. Září 2012.] http://www.orms-today.org/surveys/Simulation/Simulation6.html.
[3] Evolver 6. [Online] PALISADE. [Citace: 15. Září 2012.] http://www.palisade.com/evolver/.
[4] WITNESS Optimizer. [Online] LANNER. [Citace: 16. Září 2012.] http://www.lanner.com/en/media/witness/optimiser.cfm.
[5] ISSOP. [Online] DUALIS. [Citace: 15. Září 2012.] http://www.sim-serv.com/tool.php?i=0&ids_string=22#.
[6] OptQuest. [Online] OptTek Systems. [Citace: 15. Září 2012.] http://www.opttek.com/OptQuest.
[7] VOTAVA, V., a další., *Simulace ve strojírenství - přednášky.* [E-book] Plzeň : Západočeská univerzita v Plzni/FST/KPV, 2006.
[8] Technologies, Rockwell Automation., *OptQuest for Arena - user´s quide.* [Uživatelský manuál] Orlando, USA : Rockwell Software, Září 2006.
[9] ZELINKA, I., *Umělá inteligence v problémech globální optimalizace.* Praha : Technická literatura BEN, 2002. ISBN 80-7300-069-5.
[10] WEISE, T., "Simple Interface for Global Optimization Algorithm." *SIGOA.* [Online] 2. Listopad 2008. [Citace: 10. Prosinec 2008.] www.sigoa.org.
[11] LAMPINEN, J. a ZELINKA, I., *New Ideas in Optimization - Mechanical Engineering Design Optimization by Differential Evolution.* London : McGraw-Hill, 1999. str. 20. Sv. I. ISBN 007-709506-5.
[12] MAŘÍK, V., ŠTĚPÁNKOVÁ, O. a LAŽANSKÝ, J., *Umělá inteligence (3).* Praha : Akademie věd České republiky, Academia Praha, 2001. Sv. III. ISBN 80-200-0472-6.
[13] HYNEK, J., *Genetické algoritmy a genetické programování.* Praha : Grada Publishing, a. s., 2008. ISBN 978-247-2695-3.
[14] TVRDÍK, J., "Evoluční algoritmy - učební texty." *Virtuální informační centrum pro doktorandy informatiky.* [Online] 2004. [Citace: 6. Únor 2008.] http://prf.osu.cz/doktorske_studium/dokumenty/Evolutionary_Algorithms.pdf.
[15] WEISE, T., "E-Book "Global Optimization Algorithms - Theory and Application" 2nd Edition." *Thomas Weise - Projects.* [Online] 26. Červen 2009. [Citace: 2. Únor 2011.] http://www.it-weise.de/projects/book.pdf .
[16] ŠTEFKA, D., "Alternativy k evolučním optimalizačním algoritmům." Praha : ČVUT Praha, Fakulta jaderná a fyzikálně inženýrská, Katedra matematiky. Vedoucí práce: Ing. RNDr. Martin Holeňa, CSc., 28. Duben 2005.
[17] TETSUYUKI, T., SETSUKO, S.,., "Constrained Optimization By Applying The α Constrained Method To The Nonlinear Simplex Mmethod With Mmutations." *IEEE Transactions on Evolutionary Computation.* [Online] 3. Říjen 2005. [Citace: 23. Červenec 2008.] http://www.chi.its.hiroshima-cu.ac.jp/~takahama/eng/papers/aSimplex-TEC2005.pdf. ISSN: 1089-778X.
[18] JAKUMEIT, J., BARTH, T., REICHWALD, J., GRAUER, M., THILO, F.,., "A grid-based parallel optimization algorithm applied to a problem in metal casting industry." místo neznámé : BIOMA 2006, 2006.
[19] VOTAVA, V. a ULRYCH, Z., "Vybrané optimalizační metody pro potřeby VVSDV, Interní grant." Plzeň : Západočeská univerzita v Plzni/FST/KPV, 2006.
[20] RAŠKA, P., "Analýza optimalizačních metod pro diskrétní simulaci výrobních systémů a výrobních procesů, Práce ke státní doktorské zkoušce." Plzeň : ZČU/FST/KPV, 2008.
[21] MONTICELLI, A. J., ROMERO, R., ASADA, E. N., "Fundamentals of Simulated Annealing." [ed.] Y., EL-SHARKAWIL, L., EL-SHARKAWIL, M. A. KWNAG. *Modern Heuristic Optimization Techniques.* IEE Press. New Jersey : John Wiley & Sons, Inc., Hoboken,, 2008, 7.
[22] MAJER, P., "Moderní metody rozvrzhování výroby, Disertační práce." Brno : Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2003. http://majer.czweb.org/scheduling/.
[23] POSPÍCHAL, J., "Evoluční algoritmy - sylabus přednášky." [Online] 25. Únor 2003. [Citace: 31. Březen 2008.] ftp://math.chtf.stuba.sk/pub/vlado/Evol_alg_MFF/CHAPT2.pdf.
[24] HOOS, H. H., Sttzle, T., STOCHASTIC LOCAL SEARCH FOUNDATIONS AND APPLICATIONS. [Online] University of BC - Canada, TU Darmstadt - Germany. [Citace: 10. Únor 2012.] http://195.251.211.91/Courses/AdvancedNeuralNetworks/Slides/aaai-04-tutorial.pdf.
[25] , "Simulovaná evoluce." *EuroEkonóm.* [Online] [Citace: 5. Duben 2011.] http://www.euroekonom.sk/download2/materialy-vs-informatika2/Problemy-a-algoritmy-PAA09evoluce1.pdf.

[26] BEYER, H. G., SCHWEFEL, H. P., "Evolution Strategies - A Comprehensive Introduction." [Online] 2002. [Citace: 4. Duben 2011.] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.2414&rep=rep1&type=pdf.
[27] WHITLEY, L. D., "The GENITOR Algorithm And Selective Pressure: Whay Rank-based Allocation Of Reproductive Trials Is Best." San Francisco, CA, USA : Kaufmann Publishers Inc., 1989. Proceedings of the 3rd International Conference on Genetic Algorithms. http://citeseer.ist.psu.edu/531140.html and http://www.cs.colostate.edu/~genitor/1989/ranking89.ps.gz. 1-5586-0066-3.
[28] WETZEL, A., "Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization." *Evaluation of the Effectiveness of Genetic Algorithms in Combinatoria Optimization - PhD thesis,.* Pittsburgh : University of Pittsburgh, 1983.
[29] LEE, S., SOAK, S., KIM, K., PARK, H., JEON, M., "Statistical properties analysis of real world tournament selection in genetic algorithms." 2008. Applied Intelligence. stránky 195-205. http://www.springerlink.com/content/amr3nq330x13u32t/fulltext.pdf [. ISSN 0924-669X.
[30] MOLNÁR, Z., "Úvod do základů vědecké práce." *Sylabus pro potřeby semináře doktorandů.* Praha : ČVUT Praha, 2005.
[31] HOŘEJŠÍ, P., "Vzdálené řízení distribuované simulace založené na High Level Architecture zaměřené na modelování virtuálních výrobních systémů s výrobními procesy v něm probíhajícími, Disertační práce." Plzeň : ZČU/FST/KPV, 2007.
[32] POHLHEIM, H., "GEATbx: Example Functions." [Online] Prosinec 2006. [Citace: 20. Listopad 2011.] http://www.geatbx.com/docu/fcnindex-01.html#P204_10395.
[33] STORN, R. a PRICE, P., "A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces." místo neznámé : Kluwer Academic Publishers, 1997. Journal of Global Optimization. stránky 341–359.
[34] OŠMERA, P., "Genetické algoritmy a jejich aplikace - využití biologických a fyzikálně-informačních principů evoluce, Habilitační práce." Brno : VUT Brno, Fakulta strojního inženýrství, 2001.
[35] MITCHELL, M., *An Introduction to Genetic Algortihms.* Cambridge : MA: MIT Press, 1996.
[36] MANLIG, F., "Počítačová simulace diskrétních událostí." *MM průmyslové spektrum .* 1999.
[37] LAW, A. M., *Simulation Modeling and Analysis (Industrial Engineering and Management Science Series).* New York : McGraw-Hill Science/Engineering/Math, 2007. str. 670. Sv. IV. ISBN 978-0070592926.
[38] KŘÍŽ, P., "Ověřování optimalizačních algoritmů v úlohách diskrétní simulace se zaměřením na strojírenskou výrobu, Diplomová práce." Plzeň : Západočeská univerzita v Plzni/FST/KPV, 2003.
[39] KLEKNER, J. a RAŠKA, P., "A A Multi Agent System as a support tool for the modeling of a Supply Chain in Virtual Enterprise." Plzeň : Západočeská univerzita, 2006. Průmyslové inženýrství 2006. stránky 67-74. ISBN 80-7043-507-0.
[40] TAYLOR, D., Random Numbers. *Logical Genetics.* [Online] [Citace: 2. Září 2009.] http://logicalgenetics.com/showarticle.php?topic_id=893.
[41] BLICKLE, T., THIELE L., A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation.* [Online] CITESEER, 1996. [Citace: 24. 08 2007.] str. 361-394. http://citeseer.ist.psu.edu/blickle97comparison.html.
[42] DLOUHÝ, M., a další., *Simulace podnikových procesů.* Brno : Computer Press, 2007. ISBN 978-80-251-1649-4.
[43] MOLGA, M., SMUTNICKI, C., "Test functions for optimization needs." *Test functions for optimization needs.* [Online] 3. Květen 2005. [Citace: 20. Leden 2012.] http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.
[44] STREDA, M., "Ověřování optimalizačních metod pro diskrétní simulaci výrobních systémů a výrobních procesů." [Diplomová práce]. Plzeň : Západočeská univerzita v Plzni/FST/KPV, 2009.

18 Přehled publikační činnosti

18.1 Publikované práce

[1] RAŠKA, P. a ULRYCH, Z., "Simulation Optimization In Manufacturing Systems." Zadar : DAAAM, 2012. stránky 221-224. ISBN 978-3-901509-91-9. ISSN 2304-1382.

[2] RAŠKA, P. a ULRYCH, Z., "Implementace vybraných heuristických metod pro řízení simulačních experimentů." Plzeň : Západočeská univerzita - FST- KPV, 2011. Modelování a optimalizace podnikových procesů 2011. stránky 1-8. ISBN 978-80-261-0060-7.

[3] RAŠKA, P. a ULRYCH, Z., "Simulační optimalizace: vybrané heuristické metody a jejich aplikace." Liberec : TUV Liberec, 2011. Manufacturing systems today and tomorrow. stránky 15-28. ISBN 978-80-7372-774-1.

[4] RAŠKA, P., Analýza optimalizačních metod pro diskrétní simulaci výrobních systémů a výrobních procesů. Plzeň : Západočeská univerzita, 2008. Rigorózní práce.

[5] ŠIMON, M., RAŠKA, P. a ŠRAJER, V., "Zvyšování výkonnosti klastrů využitím pokročilých metod řízení." Zlín : Univerzita Tomáše Bati, 2008. stránky 1-7. ISBN 978-80-7318-755-2.

[6] RAŠKA, P.; HOŘEJŠÍ, P. Návrh optimalizačního modulu pro prostředí paralelní simulace In Advanced simulation of systems. Ostrava : MARQ, 2008. ISBN

[7] HOŘEJŠÍ, P.; RAŠKA, P. A SOFTWARE TOOL FOR PARALLEL SIMULATION OPTIMIZATION In MITIP 2008. Prague, 2008. ISBN

[8] RAŠKA, P.; HOŘEJŠÍ, P. AN ARCHITECTURE PROPOSAL FOR PARALLEL SIMULATION OPTIMIZATION. Prague, 2008. ISBN

[9] RAŠKA, P.;ŠIMON, M., ŠKODÁKOVÁ, P. Optimalizace výrobních systémů v síti podniků,In Digitálny podnik 2008, Žilina 2008. ISBN

[10] ULRYCH, Z.; RAŠKA, P.; HOŘEJŠÍ, P.; CANDROVÁ, K. Basic methodology for a simulation case study using parallel discrete event simulation. In MITIP 2007. Florence : Florence University, 2007. s. 347-354. ISBN 97-88-87544-107-0.

[11] EDL, M.; HOŘEJŠÍ, P.; ULRYCH, Z.; RAŠKA, P.; CANDROVÁ, K. Praktické řešení aplikace paralelní simulace. In Výrobní systémy dnes a zítra. Liberec : Technická univerzita, 2007. s. 1-5. ISBN 978-80-7372-296-2.

[12] EDL, M.; ULRYCH, Z.; HOŘEJŠÍ, P.; RAŠKA, P.; CANDROVÁ, K. Praktické řešení problematiky paralelní simulace. In Priemyselné inžinierstvo '07. Košice : Technická univerzita, 2007. s. 81-90. ISBN 978-80-8073-895-2.

[13] ULRYCH, Z.; RAŠKA, P.; HOŘEJŠÍ, P.; CANDROVÁ, K. Základní metodika simulační studie při využití paralelní diskrétní simulace. In Witness 2007. Brno : Humusoft, 2007. s. 45-58. ISBN 978-80-214-3432-5. .

[14] RAŠKA, P. MIGEN Project - use of MAS for production planning. In Soutěžní přehlídka studentských a doktorských prací FST 2007. Plzeň : Západočeská univerzita, 2007. s. 165-173. ISBN 978-80-7043-544-1.

[15] KLEKNER, J.; RAŠKA, P.; TONELLI, F.; VOTAVA, V. Řízení procesu dodavatelsko-odběratelského řetězce s ohledem na minimalizaci nákladů pomocí multiagentní simulace. In Finance a výkonnost firem ve vědě, výuce a praxi. Zlín : Univerzita Tomáše Bati, 2007. s. 1-9. ISBN 978-80-7318-536-7.

[16] KLEKNER, J.; RAŠKA, P.; TONELLI, F. Využití multi-agentního systému pro podporu modelování a řízení procesů ve virtuálním podniku. In MOPP 2007. V Plzni : Západočeská univerzita, 2007. s. 97-103. ISBN 978-80-7043-535-9.

[17] VOTAVA, V.; RAŠKA, P.; KLEKNER, J. Pokrytí výuky Simulace ve strojírenství na FST/ZČU pomocí e Learningových kurzů. In Modelling and Simulation of Systems . Ostrava : MARQ, 2007. s. 257-262. ISBN 978-80-86840-30-7.

[18] KLEKNER, J.; RAŠKA, P. A Multi Agent System as a support tool for the modeling of a Supply Chain in Virtual Enterprise. In Průmyslové inženýrství 2006. Plzeň : Západočeská univerzita , 2006. s. 67-74. ISBN 80-7043-507-0.

[19] VOTAVA, V.; ULRYCH, Z.; RAŠKA, P. E-learningová podpora výuky v oboru Průmyslové inženýrství a management. In Setkání kateder průmyslového inženýrství. Zlín : Univerzita Tomáše Bati , 2005. s. 1-5. ISBN 80-7318-373-0.

[20] VOTAVA, V.; ULRYCH, Z.; RAŠKA, P. E-learningová podpora výuky simulace ve strojírenství. In Advanced simulation of systems. Ostrava : MARQ, 2005. s. 80-85. ISBN 80-86840-16-6.

[21] RAŠKA, P.; VOTAVA, V. Projekt tvorby multimediálních kurzů simulace ve strojírenství. In PRONT 05. "PROject ManagemeNT". Plzeň : EVIDA , 2005. s. 167-178. ISBN 80-86596-65-6.

18.2 Řešené projekty

- Autorská tvorba modulu simulační optimalizace - „Životní cyklus výrobku v prostředí digitálního podniku“ - ZIVDIG r. č.: CZ.1.07/2.2.00/15.0397, 2011-2012
- Autorská tvorba modulu „Modelování a simulace a DP“ - činnost pro projekt OP VK č.CZ.1.0712.3.00/09.0163 – VYZTYMDP, 2010-2012
- GAČR - projekt 402/08/H051 – Optimalizace multidisciplinárního navrhování a modelování výrobního systému virtuálních firem, 2008-2009
- Časová racionalizace ve společnosti HP Pelzer, k.s., Křimice - měření a redukce materiálového odpadu, 2008
- Spolupráce na optimalizačním modulu v prostředí PAARE (Parallel ARENA) – řízení simulačních experimentů pomocí paralelní simulace s využitím různých optimalizačních algoritmů, 2008
- Interní grant ZČU/FST/KPV 2007 - optimalizace parametrů vstupních hodnot pro řízení experimentů paralelní simulace, 2007
- Spoluřešitel projektu simulace pro Automotive Lighting CZ – Jihlava, 2005-2006
- Interní grant ZČU/FST/KPV 2006 - spoluřešitel tvorby modelu pro paralelní simulaci, 2006
- Spoluřešitel MIGEN Project - využití Multiagentní simulace - zahraniční stáž na DIPTEM University of Genoa, 2006

18.3 e-Book

- ULRYCH, Z.; RAŠKA, P. VYZTYMDP - Modelování a simulace a DP – simulační optimalizace. Plzeň : Západočeská univerzita, 2012. ISBN 978-80-87539-15-6.
- ULRYCH, Z.; RAŠKA, P. Životní cyklus výrobku – modul optimalizace. Plzeň : Západočeská univerzita, 2012. (ISBN bude přiřazeno).
- KOPEČEK, P.; HOŘEJŠÍ, P.; RAŠKA, P. Technická informatika. Plzeň : Západočeská univerzita, 2012. (ISBN bude přiřazeno).
- VOTAVA, V.; ULRYCH, Z.; RAŠKA, P.; HOŘEJŠÍ, P. Simulace ve strojírenství. Plzeň : Západočeská univerzita, 2008. ISBN 978-80-7043-659-2.
- ULRYCH, Z.; RAŠKA, P. Simulace ve strojírenství - cvičení. Plzeň : Západočeská univerzita, 2008. ISBN 978-80-7043-662-2.
- ULRYCH, Z.; RAŠKA, P. ARENA Simulation System. Plzeň : Západočeská univerzita, 2008. ISBN 978-80-7043-662-2.
- KOPEČEK, P.; HOŘEJŠÍ, P.; RAŠKA, P. Technická informatika. Plzeň : Západočeská univerzita, 2008. ISBN 978-80-7043-656-1.

18.4 Publikovaný software

- HOŘEJŠÍ, P.; RAŠKA, P. PAARE - Parallel ARENA - řízení simulačních experimentů pomocí paralelní simulace s využitím různých optimalizačních algoritmů. Plzeň : Západočeská univerzita, 2009

ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA STROJNÍ

Studijní program: P2301 Strojní inženýrství
Studijní obor: 2301V007 Průmyslové inženýrství a management

PŘÍLOHA DISERTAČNÍ PRÁCE

OPTIMALIZAČNÍ METODY PRO DISKRÉTNÍ SIMULACI VÝROBNÍCH SYSTÉMŮ A VÝROBNÍCH PROCESŮ VE STROJÍRENSTVÍ

Autor: **Ing. Pavel Raška**

Vedoucí práce: **doc. Ing. Václav Votava, CSc.**

Akademický rok 2012/2013

Obsah přílohy

Seznam obrázků 5
Seznam tabulek 6
Seznam algoritmů 7
1 Seznam 8
2 Stochastické simulované žíhání (*Stochastic Simulated Annealing*) 9
3 Evoluční výpočetní techniky 10
3.1 Přiřazení fitness 10
3.2 Selekce 13
3.2.1 Zkrácený výběr (Truncation Selection) 13
3.2.2 Náhodná selekce (*Random Selection*) 14
3.2.3 Selekce přímo úměrná fitness (*Fitness Proportionate Selection*) - Selekce pomocí ruletového mechanismu (*Roulette Wheel Selection*) 15
3.2.4 Turnajová selekce (*Tournament Selection*) 18
3.2.5 Selekce na základě uspořádání (*Ordered Selection*) 22
4 Vývoj aplikace simulační optimalizace 23
4.1 Implementované optimalizační metody 23
4.1.1 Náhodné prohledávání - Random Search 23
4.1.2 Downhill Simplex 27
4.1.3 Stochastický horolezecký algoritmus - Hill Climbing 29
4.1.4 Stochastické zakázané prohledávání - Tabu Search 31
4.1.5 Stochastické simulované žíhání - Simulated Annealing 32
4.1.6 Stochastické lokální prohledávání - Local Search 33
4.1.7 Evoluční strategie - Evolution Strategy 34
4.1.8 Diferenciální evoluce - Differential Evolution 36
4.2 Popis prostředí softwarové aplikace simulační optimalizace 39
4.2.1 Vstupní parametry 39
4.2.2 Nastavení optimalizace 40
4.3 Softwarová aplikace Vizualizace 42
5 Simulační modely 43
5.1 Simulační model „De Jong“ 43
5.1.1 Popis simulačního modelu 43
5.1.2 Rozhodovací proměnné – vstupní parametry simulačního modelu 43
5.1.3 Účelová funkce 44
5.1.4 Kritérium ukončení 45
5.1.5 Čas simulačního experimentu 45
5.2 Simulační model „Rosenbrock“ 46
5.2.1 Popis simulačního modelu 46
5.2.2 Rozhodovací proměnné 46
5.2.3 Účelová funkce 46
5.2.4 Kritérium ukončení 47

5.2.5 Simulační experimenty 47
5.3 Simulační model „Michalewicz“ 48
5.3.1 Popis simulačního modelu 48
5.3.2 Rozhodovací proměnné 48
5.3.3 Účelová funkce 48
5.3.4 Kritérium ukončení 49
5.3.5 Simulační experimenty 49
5.4 Simulační model „Ackley“ 50
5.4.1 Popis simulačního modelu 50
5.4.2 Rozhodovací proměnné – vstupní parametry simulačního modelu 50
5.4.3 Účelová funkce 50
5.4.4 Kritérium ukončení 51
5.4.5 Simulační experiment 52
5.5 Simulační model „Doprava“ 52
5.5.1 Popis simulačního modelu 52
5.5.1.1 Doprava malých dílců pro montážní linky a pro předmontáže v předvýrobě 52
5.5.1.2 Doprava velkých dílců pro montážní linky 53
5.5.1.3 Expedice hotových výrobků od montážních linek. 53
5.5.2 Rozhodovací proměnné 56
5.5.3 Účelová funkce 56
5.5.4 Kritérium ukončení 58
5.5.5 Simulační experimenty 58
5.6 Simulační model „Penalizace“ 59
5.6.1 Popis modelu 59
5.6.2 Rozhodovací proměnné 59
5.6.3 Účelová funkce 59
5.6.4 Kritérium ukončení 62
5.6.5 Simulační experimenty 62
5.7 Simulační model „VyrobniLinka“ 63
5.7.1 Popis simulačního modelu 63
5.7.2 Rozhodovací proměnné – vstupní parametry simulačního modelu 63
5.7.3 Účelová funkce 64
5.7.4 Kritérium ukončení 65
5.7.5 Simulační experimenty 65
6 Vhodné nastavení parametrů jednotlivých optimalizačních algoritmů na základě provedených sérií 66
6.1 Random Search 66
6.1.1 Generovat stejné prvky 66
6.2 Downhill Simplex 66
6.2.1 Reflexe 66
6.2.2 Expanze 67

6.2.3 Kontrakce 67
6.2.4 Redukce 68
6.3 Hill Climbing 68
6.3.1 Rozptyl 68
6.3.2 Velikost populace 69
6.4 Tabu Search 70
6.4.1 Rozptyl 70
6.4.2 Velikost populace 70
6.4.3 Délka zakázaného seznamu 71
6.4.4 Optimalizace podle poslední populace 71
6.5 Simulated Annealing 72
6.5.1 Rozptyl 72
6.5.2 Měnit jen jeden parametr 72
6.5.3 Beta 73
6.5.4 Minimální teplota 73
6.5.5 Generovat dle rozptylu 74
6.5.6 Snižovat teplotu jen při přijetí horšího řešení 74
6.6 Local Search 75
6.6.1 Rozptyl 75
6.7 Evolution Strategy 75
6.7.1 Velikost populace 75
6.7.2 Mmp – počet potomků 76
6.7.3 q – počet předcházejících úspěchů 76
6.7.4 k – počet jedinců v turnaji 77
6.8 Differential Evolution 77
6.8.1 Velikost populace 77
6.8.2 Koef_F – koeficient adaptivního pravidla 78
6.8.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných 78

Seznam obrázků

Obr. 4-1 Snímek prostředí pro specifikaci rozhodovacích proměnných 39
Obr. 4-2 Snímek prostředí pro specifikaci omezení 39
Obr. 4-3 Snímek prostředí pro specifikaci účelové funkce 40
Obr. 4-4 Snímek prostředí pro specifikaci minimalizace/maximalizace účelové funkce a nastavení ukončení ... 40
Obr. 4-5 Snímek prostředí pro výběr optimalizačního algoritmu a nastavení jeho parametrů 41
Obr. 4-6 Snímek prostředí záložky „Spuštění optimalizace“ 41
Obr. 4-7 Snímek záložky „Spuštění série optimalizací“ 42
Obr. 4-8 Snímek softwarové aplikace Vizualizace optimalizačního experimentu – optimalizační algoritmus Evoluční strategie – simulační model „VyrobniLinka“ 42
Obr. 4-9 Snímek softwarové aplikace Vizualizace optimalizačního experimentu – nastavení grafu účelové funkce – optimalizační algoritmus Evoluční strategie – simulační model „Ackley“ 43
Obr. 5-1 Průběh účelové funkce simulačního modelu – De Jongovi funkce 45
Obr. 5-2 Průběh účelové funkce simulačního modelu – Rosenbrockovi funkce 47
Obr. 5-3 Průběh účelové funkce simulačního modelu – Michalewiczovi funkce 49
Obr. 5-4 Průběh účelové funkce simulačního modelu – Ackleyho funkce 51
Obr. 5-5 Sklad – doprava malých dílců 54
Obr. 5-6 Sklad – doprava velkých dílců 55
Obr. 5-7 Schéma modelované výrobní haly 55
Obr. 5-8 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 3; 57
Obr. 5-9 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 8; 57
Obr. 5-10 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 15; 58
Obr. 5-11 Podmínky penalizace - nedodržení stanovené průběžné doby výroby [44] 60
Obr. 5-12 Průběh účelové funkce modelu penalizace 62
Obr. 5-13 Schéma výrobní linky 63
Obr. 5-14 - Průběh účelové funkce modelu výrobní linky 65
Obr. 6-1 Úspěšnost sérií s hodnotou parametru „Generovat stejné prvky“ - Random Search 66
Obr. 6-2 Úspěšnost sérií s hodnotou parametru „Reflexe“ – Downhill Simplex 66
Obr. 6-3 Úspěšnost sérií s hodnotou parametru „Expanze“ - Downhill Simplex 67
Obr. 6-4 Úspěšnost sérií s hodnotou parametru „Kontrakce“- Downhill Simplex 67
Obr. 6-5 Úspěšnost sérií s hodnotou parametru „Redukce“- Downhill Simplex 68
Obr. 6-6 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Hill Climbing 68
Obr. 6-7 Horní pohled na povrch účelové funkce simulačního modelu „Penalizace“ 69
Obr. 6-8 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Hill Climbing 69
Obr. 6-9 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Tabu Search 70
Obr. 6-10 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Tabu Search 70
Obr. 6-11 Úspěšnost sérií s hodnotou parametru „Tabu Length“ - Tabu Search 71
Obr. 6-12 Úspěšnost sérií s hodnotou parametru „Optimalizace podle poslední populace“ - Tabu Search 71
Obr. 6-13 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Simulated Annealing 72
Obr. 6-14 Úspěšnost sérií s hodnotou parametru „Měnit jen jeden parametr“ - Simulated Annealing 72
Obr. 6-15 Úspěšnost sérií s hodnotou parametru „Beta“ - Simulated Annealing 73
Obr. 6-16 Úspěšnost sérií s hodnotou parametru „Minimální teplota“ - Simulated Annealing 73
Obr. 6-17 Úspěšnost sérií s hodnotou parametru „Generovat dle rozptylu“ - Simulated Annealing 74
Obr. 6-18 Úspěšnost sérií s hodnotou parametru „Snižovat teplotu jen při přijetí horšího řešení“ - Simulated Annealing 74
Obr. 6-19 Úspěšnost sérií s hodnotou parametru „Rozptyl“ – Local Search 75
Obr. 6-20 Úspěšnost sérií s hodnotou parametru „Velikost populace“ – Evolution Strategy 75
Obr. 6-21 Úspěšnost sérií s hodnotou parametru „Mmp“ - Evolution Strategy 76
Obr. 6-22 Úspěšnost sérií s hodnotou parametru „q“ - Evolution Strategy 76
Obr. 6-23 Úspěšnost sérií s hodnotou parametru „k“ - Evolution Strategy 77
Obr. 6-24 Úspěšnost sérií s hodnotou parametru „VelikostPopulace“ – Differential Evolution 77
Obr. 6-25 Úspěšnost sérií s hodnotou parametru „Koef_F“ - Differential Evolution 78
Obr. 6-26 Úspěšnost sérií s hodnotou parametru „Koef_C“ - Differential Evolution 78

Seznam tabulek

Tabulka 5-1 Specifikace rozhodovacích proměnných v prohledávaném prostoru 43
Tabulka 5-2 Specifikace prohledávaného prostoru 44
Tabulka 5-3 Specifikace účelové funkce 44
Tabulka 5-4 Globální minimum a maximum účelové funkce v prohledávaném prostoru 44
Tabulka 5-5 Specifikovaná kritéria ukončení 45
Tabulka 5-6 Čas potřebný k proběhnutí simulačního experimentu 45
Tabulka 5-7 Specifikace rozhodovacích proměnných v prohledávaném prostoru 46
Tabulka 5-8 Specifikace prohledávaného prostoru 46
Tabulka 5-9 Specifikace účelové funkce 46
Tabulka 5-10 Globální minimum a maximum účelové funkce v prohledávaném prostoru 46
Tabulka 5-11 Specifikovaná kritéria ukončení 47
Tabulka 5-12 Čas potřebný k proběhnutí simulačního experimentu 47
Tabulka 5-13 Specifikace rozhodovacích proměnných v prohledávaném prostoru 48
Tabulka 5-14 Specifikace prohledávaného prostoru 48
Tabulka 5-15 Specifikace účelové funkce 48
Tabulka 5-16 Globální minimum a maximum účelové funkce v prohledávaném prostoru 49
Tabulka 5-17 Specifikovaná kritéria ukončení 49
Tabulka 5-18 Čas potřebný k proběhnutí simulačního experimentu 49
Tabulka 5-19 Specifikace rozhodovacích proměnných v prohledávaném prostoru 50
Tabulka 5-20 Specifikace prohledávaného prostoru 50
Tabulka 5-21 Specifikace účelové funkce 50
Tabulka 5-22 Globální minimum a maximum účelové funkce v prohledávaném prostoru 51
Tabulka 5-23 Specifikovaná kritéria ukončení 51
Tabulka 5-24 Čas potřebný k proběhnutí simulačního experimentu 52
Tabulka 5-25 Specifikace rozhodovacích proměnných v prohledávaném prostoru 56
Tabulka 5-26 Specifikace prohledávaného prostoru 56
Tabulka 5-27 Specifikace účelové funkce 56
Tabulka 5-28 Globální minimum a maximum účelové funkce v prohledávaném prostoru 57
Tabulka 5-29 Specifikovaná kritéria ukončení 58
Tabulka 5-30 Parametry běhu na simulačním modelu 58
Tabulka 5-31 Čas potřebný k proběhnutí simulačního experimentu 58
Tabulka 5-32 Specifikace rozhodovacích proměnných v prohledávaném prostoru 59
Tabulka 5-33 Specifikace prohledávaného prostoru 59
Tabulka 5-34 Specifikace účelové funkce 61
Tabulka 5-35 Globální minimum a maximum účelové funkce v prohledávaném prostoru 61
Tabulka 5-36 Specifikovaná kritéria ukončení 62
Tabulka 5-37 Parametry běhu na simulačním modelu 62
Tabulka 5-38 Čas potřebný k proběhnutí simulačního experimentu 62
Tabulka 5-39 Specifikace rozhodovacích proměnných v prohledávaném prostoru 63
Tabulka 5-40 Specifikace prohledávaného prostoru 64
Tabulka 5-41 Specifikace účelové funkce 64
Tabulka 5-42 Globální minimum a maximum účelové funkce v prohledávaném prostoru 64
Tabulka 5-43 Specifikovaná kritéria ukončení 65
Tabulka 5-44 Parametry běhu na simulačním modelu 65
Tabulka 5-45 Čas potřebný k proběhnutí simulačního experimentu 65

Seznam algoritmů

Algoritmus 3-1 AssignFitnessPrevalence1 10
Algoritmus 3-2 AssignFitnessPrevalence2 [10] 11
Algoritmus 3-3 AssignFitnessRank [10] 12
Algoritmus 3-4 AssignFitnessTournament [10] 13
Algoritmus 3-5 TruncationSelect [15] 13
Algoritmus 3-6 CFfX - Komparace hodnot fitness funkce dvou prvků v případě minimalizace fitness funkce 14
Algoritmus 3-7 RandomSelect$r$ [15] 14
Algoritmus 3-8 RandomSelect$w$ [15] 15
Algoritmus 3-9 RouletteWheelSelect$r$ 17
Algoritmus 3-10 RouletteWheelSelect$w$ 18
Algoritmus 3-11 TournamentSelect$r$ [15] 19
Algoritmus 3-12 TournamentSelect$w1$ [15] 20
Algoritmus 3-13 TournamentSelect$w2$ [15] 21
Algoritmus 3-14 TournamentSelectr, p [15] 21
Algoritmus 3-15 OrderedSelect$r$ [15] 22
Algoritmus 3-16 OrderedSelect$w$ [15] 23
Algoritmus 4-1 RandomSearch 24
Algoritmus 4-2 Create - generování prvku 24
Algoritmus 4-3 RandomSearch s využitím množiny optim 25
Algoritmus 4-4 UpdateOptimalSet [10] 25
Algoritmus 4-5 CFFX - Komparace hodnot účelové funkce dvou prvků v případě minimalizace účelové funkce 26
Algoritmus 4-6 ExtractOptimalSet [10] 26
Algoritmus 4-7 DownhillSimplex [15] 28
Algoritmus 4-8 CreatePop - generování množiny prvků 28
Algoritmus 4-9 StochasticHillClimbing 29
Algoritmus 4-10 Mutate$u$ - Transformace složek prvku na základě rovnoměrného rozdělení 30
Algoritmus 4-11 Perturbation - Perturbace souřadnic prvku mimo meze [14] 30
Algoritmus 4-12 StochasticTabuSearch 32
Algoritmus 4-13 StochasticSimulatedAnnealing 33
Algoritmus 4-14 StochasticLocalSearch 33
Algoritmus 4-15 ES$\mu + \lambda$ 35
Algoritmus 4-16 Mutate$n$- transformace složek prvku na základě normálního rozdělení 36
Algoritmus 4-17 DE 37
Algoritmus 4-18 DEBest 38
Algoritmus 5-1 Výše penalizace u výrobku 61

1 Seznam

Následující metody umožňují přidání prvku do seznamu, odstranění prvku ze seznamů, třídění seznamů nebo vyhledávání uvnitř seznamů atd.:

- **Length** ($n = \mathrm{Length}(L)$) funkce která navrací délku seznamu L tj. počet prvků v seznamu, ($n = \mathrm{Length}(L) \equiv |L|$).
- **CreateList** ($L = \mathrm{CreateList}(n, x)$) funkce pro vytvoření nového seznamu L o délce *n*, naplněný prvkem x. Jestliže je vytvořen seznam o délce 0, parametr x může být vynechán ($L = \mathrm{CreateList}(0,0) \equiv (\ )$).
- **InsertListItem** ($M = \mathrm{InsertListItem}(L, i, x)$) funkce vytvoření nového seznamu M pomocí vložení jednoho prvku *x* do seznamu L s indexem *i* ($\forall i: 0 \leq i \leq \mathrm{Length}(L)$). Tím se posunou všechny prvky seznamu umístěné za tímto indexem o jednu pozici doprava.

$$M = \mathrm{InsertListItem}(L, i, x) \Leftrightarrow \mathrm{Length}(M) = \mathrm{Length}(L) + 1 \wedge M[i] = x \wedge \forall j: 0 \leq j < i \Rightarrow M[j] = L[j] \quad \forall j: i \leq j < \mathrm{Length}(L) \Rightarrow M[j+1] = L[j] \tag{1.1}$$

- **AddListItem** ($M = \mathrm{AddListItem}(L, x)$) funkce pro vložení jednoho prvku na konec seznamu.

$$\mathrm{AddListItem}(L, x) \equiv \mathrm{InsertList}(L, \mathrm{Length}(L), x) \tag{1.2}$$

- **AppendList** ($M = \mathrm{AppendList}(L_1, L_2)$) funkce pro vytvoření nového seznamu *M*, který vznikne přidáním všech prvků ze seznamu L_2 do seznamu L_1. Rekurzivně lze definovat:

$$\mathrm{AppendList}(L_1, L_2) \equiv \begin{cases} L_1 \ \text{ jestli } \mathrm{Length}(L_2) = 0 \\ \mathrm{AppendList}(\mathrm{AddListItem}(L_1, L_2[0]), \\ \qquad \mathrm{DeleteListItem}(L_2, 0)) \ \text{ jinak} \end{cases} \tag{1.3}$$

- **DeleteListItem** ($M = \mathrm{DeleteListItem}(L, i)$) funkce pro vytvoření nového seznamu *M* pomocí odstranění prvku *x* na pozici *i* ($\forall i: 0 \leq i < \mathrm{Length}(L) - 1$) ze seznamu *L* ($\mathrm{Length}(L) \geq i + 1$).

$$M = \mathrm{DeleteListItem}(L, i) \Leftrightarrow \mathrm{Length}(M) = \mathrm{Length}(L) - 1 \wedge \forall j: 0 \leq j < i \Rightarrow M[j] = L[j] \quad \forall j: i < j < \mathrm{Length}(L) \Rightarrow M[j-1] = L[j] \tag{1.4}$$

- **DeleteListRange** ($M = \mathrm{DeleteListRange}(L, i, c)$) funkce pro vytvoření nového seznamu *M* pomocí odstranění *c* prvků začínající indexem *i* ($\forall i: 0 \leq i < \mathrm{Length}(L)$ ze seznamu *L* ($\mathrm{Length}(L) \geq i + c$).

$$M = \mathrm{DeleteListRange}(L, i, c) \Leftrightarrow \mathrm{Length}(M) = \mathrm{Length}(L) - c \wedge \forall j: 0 \leq j < i \Rightarrow M[j] = L[j] \quad \forall j: i + c \leq j < \mathrm{Length}(L) \Rightarrow M[j+c] = L[j] \tag{1.5}$$

- **CountItemOccurrences** ($y = \mathrm{CountItemOccurrences}(x, L)$) funkce, která vrací počet výskytů prvku *x* v seznamu *L*.

$$\mathrm{CountItemOccurrences}(x, L) = |\{i \in 0 \ldots \mathrm{Length}(L) - 1: L[i] = x\}| \tag{1.6}$$

- **SubList** ($M = \mathrm{SubList}(L, i, c)$) funkce vrací nový seznam *M* vyjmutých *c* prvků ze seznamu *L* začínající indexem *i*.

$$\mathrm{SubList}(L, i, c) \equiv \mathrm{DeleteListRange}(\mathrm{DeleteListRange}(l, 0, i), c, \mathrm{Length}(L) - i - c) \tag{1.7}$$

- **CF** - komparační (porovnávací) funkce – ($\mathrm{CF}(l_1, l_2)$), která vrací zápornou hodnotu jestli $l_1 < l_2$, kladnou hodnotu když $l_1 > l_2$ a hodnotu 0, pokud $l_1 = l_2$. Funkce je využita pro párové porovnání dvou prvků ze seznamu při třídění. Pokud bude u komparační funkce použito indexování, znamená to, že prvky byly porovnány na základě hodnot funkce vypočtené pro každý prvek např. $\mathrm{CF}_{F(\mathbf{X})}(l_1, l_2)$ ... prvky byly porovnány na základě hodnot účelové funkce u každého prvku.
- **Sort** $M = \mathrm{Sort}_{\mathrm{a}}(L, \mathrm{CF})$ a $M = \mathrm{Sort}_{\mathrm{d}}(L, \mathrm{CF})$) často je užitečné používat setříděný seznam, proto je zde definována funkce, která třídí seznam *L* podle pořadí buď vzestupně – ascending - $M = \mathrm{Sort}_{\mathrm{a}}(L, \mathrm{CF})$ nebo sestupně – descending - $M = \mathrm{Sort}_{\mathrm{d}}(L, \mathrm{CF})$ pomocí komparační funkce.

Pro vzestupné třídění lze definovat:

$$M = \mathrm{Sort_a}(L, \mathrm{CF}) \tag{1.8}$$

$$\forall l \in L \exists i \in [0, \mathrm{Length}(L) - 1]: M[i] = l \tag{1.9}$$

$$\mathrm{Length}(M) = \mathrm{Length}(L) \tag{1.10}$$

$$\forall i: 0 \leq i < \mathrm{Length}(L) \Rightarrow \mathrm{CF}(M[\mathrm{i}], M[\mathrm{i}+1]) \leq 0 \tag{1.11}$$

U sestupného třídění je rozdíl pouze v rovnici (1.11):

$$M = \mathrm{Sort_d}(L, \mathrm{CF}) \tag{1.12}$$

$$\forall i: 0 \leq i < \mathrm{Length}(L) \Rightarrow \mathrm{CF}(M[\mathrm{i}], M[\mathrm{i}+1]) \geq 0 \tag{1.13}$$

- $\mathbf{Search_u}(\boldsymbol{l}, \boldsymbol{L})$ - hledání prvku *l* v nesetříděném seznamu *L* znamená projít všechny prvky do té doby, dokud není prvek nalezen, nebo dokud se nenarazí na konec seznamu.

$$\mathrm{Search_u}(l, L) = \begin{cases} i: L[i] = l & \text{jestli } \ l \in L \\ -1 & \text{jinak} \end{cases} \tag{1.14}$$

- **RemoveListItem** ($M = \mathrm{RemoveListItem}(L, x)$) nalezení prvního výskytu prvku *x* v seznamu *L* pomocí vhodného algoritmu a tento prvek bude vymazán (navrácení nového seznamu *M*).

$$M = \mathrm{RemoveListItem}(L, x) \Leftrightarrow \begin{cases} L & \text{jestli } \mathrm{Search}(x, L) < 0 \\ \mathrm{DeleteListItem}\big(L, \mathrm{Search}(x, L)\big) & \text{jinak} \end{cases} \tag{1.15}$$

2 Stochastické simulované žíhání (*Stochastic Simulated Annealing*)

Existuje mnoho variant jak snižovat teplotu. Vyjmenujeme alespoň některé z nich:

1. **Redukce teploty** po definovaném počtu provedených iterací. [15]

$$T^{(k+1)} \leftarrow T^{(k)} * (1 - \varepsilon) \quad jestli \ \ (k \bmod m) = 0 \tag{2.1}$$

kde:

- $T^{(k+1)}$ … Aktuální teplota.
- $T^{(k)}$ … Předchozí teplota v přechozí k-té iteraci.
- ε … Definovaná konstanta, doporučená hodnota $\varepsilon \in [0.01, 0.5]$ podle literatury [21], pokud $m = 1$.
- m … Definovaná konstanta počtu iterací, po nichž má být provedena redukce teploty.
- k … Čítač iterací.
- mod … Zbytek po celočíselném dělení.

2. **Variabilní míra ochlazení**: [21]

$$T^{(k+1)} \leftarrow \frac{T^{(k)}}{\left(1 + \frac{\ln(1+\delta) * T^{(k)}}{3 * \sigma(T^{(k)})}\right)} \tag{2.2}$$

kde:

- δ … Variabilní míra ochlazení, doporučená hodnota $\delta \in [0.01, 0.2]$.
- $\ln$ … Přirozený logaritmus.
- $\sigma\big(T^{(k)}\big)$ … Směrodatná odchylka hodnot účelové funkce prvků generovaných při předchozí teplotě $T^{(k)}$.

3. **Variabilní míra ochlazení**, která zachycuje stejný princip jako v předchozím bodu: [21]

$$T^{(k+1)} \leftarrow \frac{T^{(k)}}{e^{\left(\frac{\lambda * T^{(k)}}{\sigma(T^{(k)})}\right)}} \tag{2.3}$$

kde:

- λ … Variabilní míra ochlazení, doporučená hodnota $\lambda \leq 1.0$.
- e … Základ přirozeného logaritmu umocněný na zadaný exponent.

4. **Simulované hašení**: [16]

$$T^{(k+1)} \leftarrow \frac{T^{(k)}}{1+\beta * T^{(k)}} \tag{2.4}$$

kde:

- β … Nezáporná konstanta pro simulované hašení, $\beta \in \mathbb{R}^{+}, \beta < 1$.

3 Evoluční výpočetní techniky

3.1 Přiřazení fitness

❖ AssignFitnessPrevalence$_1$:

Hodnota fitness jedince $\mathbf{X} \in X_{\text{Pop}}$ v algoritmu (viz Algoritmus 3-1 $\text{AssignFitnessPrevalence}_\mathbf{1}$) je nepřímo úměrná počtu lepších jedinců z populace tj. $fit = \frac{1}{\text{počet lepších jedinců v populaci} +1}$ (řádek 6). K porovnání dvou jedinců je použita komparační funkce $\text{CF}_{F(\mathbf{X})}$, která porovnává dva jedince populaci = seznamu X_{Pop} na základě jejich hodnoty účelové funkce $F(\mathbf{X})$ – (řádek 5). Jedinci se stejnými hodnotami účelové funkce mají hodnotu fitness rovnu 1.
Přiřazení takto vypočtené hodnoty fitness jedinci je prováděno pomocí AssignFitnessTo (řádek 7).

AssignFitnessPrevalence$_1$ (X_{Pop})	
Procedura, která přiřazuje všem jedincům (prvkům) z populace (seznamu) X_{Pop} skalární hodnotu fitness na základě komparační funkce. Každá z těchto hodnot je nepřímo úměrná počtu lepších jedinců z populace	
Vstup:	X_{Pop}: Populace, ve které bude přiřazeno každému jedinci hodnota fitness
Data:	$\text{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnot účelové funkce u dvou aktuálně vybraných jedinců z populace X_{Pop}
Data:	i, j, cnt: Proměnné - čítače
Data:	fit: Hodnota fitness jedince, která je přiřazena jedinci

```
1   begin
2     for i ⟵ Length(X_Pop) − 1 downto 0 do begin
      (*populace transformovaná na seznam X_Pop, kde Length(X_Pop) značí velikost seznamu X_Pop*)
3       cnt ⟵ 0;
4       for j ⟵ Length(X_Pop) − 1 downto 0 do
5         if (i <> j) and CF_F(X)(X_Pop[i], X_Pop[j]) < 0 then cnt ⟵ cnt + 1;
      (*hodnota účelové funkce i-tého prvku (jedince) seznamu (populace) X_Pop[i] je lepší než X_Pop[j] a zároveň (i ≠ j)*)
6       fit ⟵ 1/(cnt+1)  //hodnota fitness
7       AssignFitnessTo(X_Pop[i], fit) ;
      (*přiřazení skalární hodnoty fit jedinci na i-té pozici v X_Pop pomocí procedury AssignFitnessTo*)
8     end;
9   end;
```

Algoritmus 3-1 AssignFitnessPrevalence$_1$

❖ AssignFitnessPrevalence$_2$:

V následujícím algoritmu (viz Algoritmus 3-2 $\text{AssignFitnessPrevalence}_\mathbf{2}$) jsou přiřazeny všem jedincům z X_{Pop}, skalární hodnoty fitness, které vyjadřují pořadí jedinců v populaci podle hodnot účelové funkce (řádek 5). Jedinci se stejnými hodnotami účelové funkce mají stejnou hodnotu fitness.

AssignFitnessPrevalence$_2(X_{\text{Pop}})$	
Procedura, která přiřazuje všem jedincům z X_{Pop} skalární hodnoty fitness úměrné pořadí jedinců v populaci, které jsou lepší než právě vybraný jedinec.	
Vstup:	X_{Pop}: Populace, ve které bude přiřazeno každému jedinci hodnota fitness
Data:	$\text{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnot účelové funkce u dvou aktuálně vybraných jedinců z populace X_{Pop}
Data:	i, j, cnt: Proměnné - čítače
Data:	fit: Hodnota fitness jedince, která je přiřazena jedinci

```
1   begin
2     for i ⟵ Length(X_Pop) − 1 downto 0 do begin
3           cnt ⟵ 0;
4           for j ⟵ Length(X_Pop) − 1 downto 0 do
5             if (i <> j) and CF_F(X)(X_Pop[j], X_Pop[i]) < 0 then cnt ⟵ cnt + 1;
6           fit ⟵ cnt;
7           AssignFitnessTo(X_Pop[i], fit);
8     end;
9   end;
```

Algoritmus 3-2 AssignFitnessPrevalence$_2$ **[10]**

❖ AssignFitnessRank:

V následujícím algoritmu (viz Algoritmus 3-3) je nejprve celá populace setříděna podle komparační funkce $\text{CF}_{F(\mathbf{X})}$ (v tomto případě podle hodnot účelové funkce – řádek 2). Jedinci je přiřazena hodnota fitness v závislosti na pořadí v setříděné populaci – prvnímu hodnota 1 (řádek 4) a dalšímu, pokud je jeho hodnota účelové funkce lepší, je přiřazeno fit o jednu vyšší než předchozímu jedinci (řádek 6). Jedinci se stejnými hodnotami účelové funkce mají přiřazenu stejnou hodnotu fitness.

AssignFitnessRank(X_{Pop})	
Procedura, která setřídí všechny jedince v populaci podle komparační funkce a přiřazuje všem jedincům z X_{Pop} skalární hodnoty fitness podle jejich pořadí v populaci	
Vstup:	X_{Pop}: Populace, ve které bude přiřazeno každému jedinci hodnota fitness
Data:	$\text{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnot účelové funkce u dvou aktuálně vybraných jedinců z populace X_{Pop}
Data:	i: Proměnná – čítač
Data:	r: Pořadí jedince
Data:	fit: Hodnota fitness jedince, která je přiřazena jedinci

```
1   begin
2     X_Pop ⟵ Sort_a(X_Pop, CF_F(X));
      (*vzestupné (maximalizace-sestupné) třídění celé populace podle komparační funkce, která
      porovnává jedince podle hodnoty účelové funkce*)
3     r ⟵ 1;
4     AssignFitnessTo(X_Pop[0], 1);
5     for i ⟵ Length(X_Pop) − 1 downto 0 do begin
6       if CF_F(X)(X_Pop[i], X_Pop[i − 1]) < 0 then r ⟵ r + 1;
      //hodnota účel. funkce prvku (jedince) seznamu X_Pop[i] je lepší než X_Pop[i − 1]
7       fit ⟵ r;
8       AssignFitnessTo(X_Pop[i], fit);
```

```
9        end;
10     end;
```

Algoritmus 3-3 AssignFitnessRank **[10]**

❖ AssignFitnessTournament:

V následujícím algoritmu (viz Algoritmus 3-4 AssignFitnessTournament) je zachycen princip soutěžení jedince proti vybraným soupeřům. Jedinec soutěží v q zápasech (řádek 3) proti r soupeřům (řádek 6). Obvykle bývá nastavena hodnota počtu soupeřů na $r = 1$. Vítězství jedince je určeno na základě komparační funkce, která porovnává hodnoty účelové funkce jedince proti soupeřově (řádek 8). Soupeř je vybrán náhodně z populace X_{Pop}.
Funkce $\mathrm{Random_u}\big(\mathrm{Length}(X_{\mathrm{Pop}})\big)$ generuje náhodné číslo indexu jedince od indexu $[0]$ tj. index prvního jedince v seznamu až do indexu $< \mathrm{Length}(X_{\mathrm{Pop}})$ tj. menšího než počet všech jedinců v populaci. Hodnota fitness je úměrná počtu vyhraných zápasů (řádek 11).

Poznámka: Závorka ⌊ ⌋ značí nejmenší celé číslo větší nebo rovné hodnotě uvnitř závorky. V případě, kdy index jedince v populaci = $\lfloor \mathrm{Random_u}\big(\mathrm{Length}(X_{\mathrm{Pop}})\big) \rfloor$ to znamená, že indexy se pohybují pouze v oblasti od 0 až do hodnoty $\mathrm{Length}\big(X_{\mathrm{Pop}}\big) - 1$. Tento zápis se např. v programovacím jazyku Delphi nahrazuje příkazem **Trunc**.
Závorka ⌈ ⌉ značí největší celé číslo větší nebo rovné hodnotě uvnitř závorky. Synonymem této závorky v programovacím jazyku Delphi je příkaz **Round.**

AssignFitnessTournament(X_{Pop})	
Procedura, kde hodnota fitness jedince je úměrná počtu vyhraných zápasů soutěžení proti vybraným soupeřům (v q zápasech proti r soupeřům)	
Vstup:	X_{Pop}: Populace, ve které bude přiřazeno každému jedinci hodnota fitness
Vstup:	r: Počet soupeřů v zápasu (obvykle $r = 1$)
Vstup:	q: Počet zápasů jedince
Data:	$\mathrm{CF}_{F(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnot účelové funkce u dvou aktuálně vybraných jedinců z populace X_{Pop}
Data:	i, j, k: Proměnné – čítače
Data:	z: Počet vyhraných zápasů
Data:	fit: Hodnota fitness jedince, která je přiřazena jedinci

```
1     begin
2       for i ⟵ Length(X_Pop) − 1 downto 0 do begin
3           z ⟵ q;    //počet zápasů;
4           for j ⟵ q downto 1 do begin
5               b ⟵ True;    //jestliže je b nabývá hodnoty True = prohra
6               k ⟵ r;    //počet protivníků v jednom kole
7               while (k > 0) and b do begin
8                   b ⟵ CF_F(X)(X_Pop[i], X_Pop[⌊Random_u(0,Length(X_Pop))⌋]) < 0;
      //prohra jedince s náhodně vybraným protivníkem z X_Pop ;
9                   k ⟵ k − 1;
10              end;
11              if b then z ⟵ z − 1;//čím více jedinec vyhraje zápasů, tím menší bude z
12          end;
13          fit ⟵ z;
14          AssignFitnessTo(X_Pop[i], fit) ;
```

```
15      end;
16    end;
```

Algoritmus 3-4 AssignFitnessTournament **[10]**

3.2 Selekce

3.2.1 Zkrácený výběr (Truncation Selection)

- TruncationSelect**:**

V následujícím algoritmu (viz Algoritmus 3-5 TruncationSelect) je použito vzestupné třídění (minimalizace fitness) vstupního seznamu jedinců X_{Pop} na základě komparační funkce $\mathrm{CF}_{\mathrm{f}(\mathbf{X})}$ (řádek 4). Komparační funkce využívá pro porovnávání hodnot fitness dvou jedinců funkce $\mathrm{f}(\mathbf{X})$. Tato funkce poskytuje hodnoty fitness, které byly přiřazeny např. pomocí některé varianty algoritmu AssignFitness a jedinec $\mathbf{X}$ je předáván ve formě i-tého indexu prvku seznamu $\mathbf{X}_i = X_{\mathrm{Pop}}[i] \Rightarrow \mathrm{CF}_{\mathrm{f}(X_{\mathrm{Pop}}[i])}, i \in [0, m-1]$.

Hodnotu odříznutí (*Cut-off*) v algoritmu lze vnímat jako výběr nejlepších „c" jedinců ze seznamu X_{Pop} – v algoritmu dochází ke kopírování jedince ze seznamu X_{Pop} umístěného na pozici $[i \bmod c]$ tolikrát dokud velikost seznamu X_{MP} určeného k páření nedosáhne hodnoty m_{MP} – řádek 6 umístěný uvnitř cyklu s pevným krokem - řádek 5.

Hodnota odříznutí c by měla být menší než velikost seznamu, ze kterého bude proveden výběr do X_{MP} (řádek 3). Velikost populace $m = \mathrm{Length}(X_{\mathrm{Pop}})$ ze které bude proveden výběr do X_{MP}, by měla být větší než velikost seznamu jedinců určených k páření $m_{\mathrm{MP}} < m$.

Běžně jsou pro m_{MP} používány hodnoty $\frac{\mathrm{Length}(X_{\mathrm{Pop}})}{2}$, nebo $\frac{\mathrm{Length}(X_{\mathrm{Pop}})}{3}$. [15]

$X_{\mathrm{MP}} \longleftarrow$ TruncationSelect($X_{\mathrm{Pop}}, m_{\mathrm{MP}}, c$)	
Funkce, která na základě hodnot fitness v seznamu X_{Pop} vybere m_{MP} jedinců a umístí je do seznamu pro reprodukci – výstup funkce.	
Vstup:	X_{Pop}: Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}: Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	c: Hodnota odříznutí (*Cut-off*)
Data:	$\mathrm{CF}_{\mathrm{f}(\mathbf{X})}$: Komparační funkce umožňující porovnání hodnot fitness u dvou aktuálně vybraných jedinců ze seznamu X_{Pop}
Data:	$\mathrm{f}(\mathbf{X})$: Funkce, která poskytuje jako výstup hodnotu fitness jedince $\mathbf{X}$
Data:	i: Proměnná - čítač
Výstup:	X_{MP}: Výsledná populace určená k páření

```
1     begin
2       X_MP ⟵ ( );
3       c ⟵ min(c, Length( X_Pop));
4       X_Pop ⟵ Sort_a(X_Pop, CF_f(X));
5       for i ⟵ 0 to m_MP − 1 do
6           X_MP ⟵ AddListItem(X_MP, X_Pop[i mod c]);
7       Result ⟵ X_MP;
8     end;
```

Algoritmus 3-5 TruncationSelect **[15]**

$l \longleftarrow \mathrm{CF}_{\mathrm{f(X)}}(\mathbf{X}_1, \mathbf{X}_2)$		
Funkce, jejímž výstupem je $\{-1\}$ v případě, že první prvek je lepší než druhý – v případě minimalizace fitness funkce, hodnota fitness funkce prvního prvku je menší než hodnota fitness funkce druhého prvku. Výstup $\{1\}$ nastává v případě, že první prvek je horší než druhý, a výstupem je$\{0\}$, pokud jsou si prvky rovny kvalitou.		
Vstup:	$\mathbf{X}_1$:	První porovnávaný prvek
Vstup:	$\mathbf{X}_2$:	Druhý porovnávaný prvek
Data:	$\mathrm{f}(\mathbf{X})$:	Fitness funkce
Výstup:	l:	Výstup komparační funkce – možné výstupy $\{-1,1,0\}$

```
1   begin
2     if f(X1) < f(X2) then result ⟵ −1        //první prvek je lepší než druhý
3     else   if f(X1) > f(X2) then result ⟵ 1 //první prvek je horší než druhý
4            else result ⟵ 0; //prvky jsou stejné
5   end;
```

Algoritmus 3-6 $\mathrm{CF}_{\mathrm{f(X)}}$ **- Komparace hodnot fitness funkce dvou prvků v případě minimalizace fitness funkce**

3.2.2 Náhodná selekce (*Random Selection*)

❖ $\mathrm{RandomSelect}_r$**:**

Náhodná selekce se substitucí - jedinec se může ve vybrané populaci k páření vyskytovat vícekrát. Podstata tohoto algoritmu je zachycena v následujícím algoritmu (viz Algoritmus 3-7 $\mathbf{RandomSelect}_r$). Do populace k páření jsou náhodně vybíráni jedinci na základě rovnoměrného rozdělení. Náhodné číslo značící index vybraného jedince se pohybuje se v rozmezí 0 až $\mathrm{Length}(X_{\mathrm{Pop}}) - 1$ (řádek 4). Kolik bude populace určená k páření obsahovat jedinců je dáno počtem iterací v cyklu for (řádek 3).

$X_{\mathrm{MP}} \longleftarrow \mathrm{RandomSelect}_r(X_{\mathrm{Pop}}, m_{\mathrm{MP}})$		
Funkce, která na základě rovnoměrného rozdělení vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Data:	i:	Proměnná - čítač
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     X_MP ⟵ ( );
3     for i ⟵ 0 to m_MP − 1 do
4       X_MP ⟵ AddListItem(X_MP, X_Pop[[Random_u(Length(X_Pop))]]);
5     Result ⟵ X_MP;
6   end;
```

Algoritmus 3-7 $\mathbf{RandomSelect}_r$ **[15]**

❖ $\mathrm{RandomSelect}_w$**:**

Náhodná selekce bez substituce - jedinec se může ve vybrané populaci k páření vyskytovat maximálně jedenkrát. Podstata tohoto algoritmu je zachycena v dalším algoritmu (viz Algoritmus 3-8). Do populace k páření jsou náhodně vybíráni jedinci na základě rovnoměrného rozdělení. Náhodné číslo značící index vybraného jedince se pohybuje se v rozmezí 0 až $\mathrm{Length}(X_{\mathrm{Pop}}) - 1$ (řádek 4). Kolik bude populace určená k

páření obsahovat jedinců je dáno počtem iterací v cyklu for (řádek 3). Vybraný jedinec je následně odstraněn z populace, ze které jsou vybíráni jedinci (řádek 6).

$X_{\mathrm{MP}} \longleftarrow \mathrm{RandomSelect}_w(X_{\mathrm{Pop}}, m_{\mathrm{MP}})$	
Funkce, která na základě rovnoměrného rozdělení vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.	
Vstup:	X_{Pop}: Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}: Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření, $m_{\mathrm{MP}} \leq \mathrm{Length}(X_{\mathrm{Pop}})$
Data:	i: Proměnná - čítač
Data:	j: Index vybraného jedince
Výstup:	X_{MP}: Výsledná populace určená k páření

```
1   begin
2     X_MP ⟵ ( );
3     for i ⟵ 0 to m_MP − 1 do begin
4       j ⟵ ⌊Random_u(Length(X_Pop))⌋;
5       X_MP ⟵ AddListItem(X_MP, X_Pop[j]);
6       X_Pop ⟵ DeleteListItem(X_MP, j);
7     end;
8     Result ⟵ X_MP;
9   end;
```

Algoritmus 3-8 $\mathbf{RandomSelect}_w$ [15]

3.2.3 Selekce přímo úměrná fitness (*Fitness Proportionate Selection*) - Selekce pomocí ruletového mechanismu (*Roulette Wheel Selection*)

- $\mathrm{RouletteWheelSelect}_r$:

V následujícím algoritmu (viz Algoritmus 3-9) je uvedena metoda ruletové selekce na základě principu selekce **se substitucí** - jedinec se může v populaci vyskytovat několikrát.

Prvním krokem algoritmu je vytvoření prázdného pole, do kterého budou ukládány hodnoty fitness jednotlivých jedinců (viz 7, řádek 2). Pole, ve kterém jsou uloženy hodnoty fitness (řádek 6), je dále použito k výpočtu normovaných hodnot fitness tak, že je nalezena nejmenší a nejvyšší hodnota fitness u jedinců ve vstupní populaci X_{Pop} (řádek 8 a 9). V případě, že by se tyto hodnoty sobě rovnaly (řádek 11), je provedena oprava tak, že k maximu bude přičtena hodnota 1 (řádek 12) a od minima bude odečtena hodnota 1 (řádek 13). Dále bude následovat normování hodnot fitness uložených v poli (řádek 17). Hodnoty fitness každého jedince uložené v poli postupně přepisujeme vypočtenými hodnotami normovaných hodnot fitness (řádek 18). Zároveň při každém výpočtu takové hodnoty je prováděno přičtení hodnoty k sumě normovaných hodnot (řádek 19), aby bylo možné spočítat pravděpodobnost výběru daného jedince (řádek 23). Ruletové kolo si lze představit jako výsečový graf, kde každý jedinec má přiřazenu výseč (řádek 25), jejíž dolní mez je rovna součtu pravděpodobností všech předchozích jedinců a horní mez výseče je dána přičtením vypočtené pravděpodobnosti k sumě všech pravděpodobností předchozích jedinců (řádek 24). Při roztočení ruletového kola se vhodí kulička, která padne do nějaké výseče. Synonymem roztočení kola a nalezení výseče je vygenerování náhodného čísla (řádek 28) a nalezení intervalu kam patří - hodnoty v poli, která je větší než náhodné číslo (řádek 29 a 30) - pokud se narazilo v poli na větší hodnotu než náhodné číslo, dojde k vystoupení z **for** cyklu pomocí příkazu **break** a tím je znán index jedince, který představuje poslední hodnota čítače cyklu. Jedinec, jemuž výseč patří, je následně zařazen do populace určené k páření (řádek 31).

$X_{\mathrm{MP}} \leftarrow \mathrm{RouletteWheelSelect}_r(X_{\mathrm{Pop}}, m_{\mathrm{MP}})$	
Funkce, která na základě ruletového mechanismu vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát.	
Vstup:	X_{Pop}: Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}: Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Data:	f: Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	i, j: Proměnné - čítače
Data:	a: Dočasná proměnná pro uložení numerické hodnoty
Data:	A: Pole hodnot kumulativních ohodnocení normovaných fitness
Data:	minf, maxf: Hodnoty minima fitness, maxima fitness,
Data:	$norm$f: Normovaná hodnota fitness i-tého jedince z X_{Pop}
Data:	$sumNormf$: Součet normovaných fitness
Data:	Pr: Pravděpodobnost výběru jedince normovaných hodnot fitness
Data:	sum: Kumulativní normované ohodnocení fitness
Výstup:	X_{MP}: Výsledná populace určená k páření

```
1   begin
2     A ← CreateList(Length(X_Pop), 0);
3     minf ← ∞;
4     maxf ← −∞;
5     for i ← 0 to Length(X_Pop) − 1 do begin
6       a ← f(X_Pop[i]);
      //ohodnocení jedince pomocí fitness
7       A[i] ← a
8       if a < minf then minf ← a;
      //nejmenší hodnota fitness
9       if a > maxf then maxf ← a;
10    end;
11    if maxf = minf then begin
      //oprava, pokud by největší hodnota fitness = nejmenší hodnotě fitness
12      maxf ← maxf + 1;
13      minf ← minf − 1;
14    end;
15    sumNormf ← 0;
16    for i ← 0 to Length(X_Pop) − 1 do begin
17      Normf ← (maxf − A[i]) / (maxf − minf);
      //normování hodnoty fitness
18      A[i] ← Normf;
19      sumNormf ← sumNormf + Normf;
20    end;
21    sum ← 0;
22    for i ← 0 to Length(X_Pop) − 1 do begin
23      Pr ← A[i] / sumNormf;
      //pravděpodobnost výběru jedince
24      sum ← sum + Pr;
      //kumulativní ohodnocení
25      A[i] ← sum;
```

```
        //vektor kumulativních ohodnocení
26      end;
27      for j ⟵ 0 to m_MP − 1 do begin
28          a ⟵ Random_u(0, sum);
29          for i ⟵ 0 to Length(X_Pop) − 1 do
30              if (A[i] > a) then break;
        //nalezení segmentu v ruletě tj. i-tého indexu jedince v populaci
31          X_MP ⟵ AddListItem(X_MP, X_Pop[i]);
        //výběr i -tého jedince do populace určené k páření
32      end;
33      Result ⟵ X_MP;
34  end;
```

Algoritmus 3-9 RouletteWheelSelect$_r$

❖ RouletteWheelSelect$_w$:

Princip ruletové selekce, ale tentokrát bez substituce (viz Algoritmus 3-10), je uveden v dalším algoritmu. V tomto algoritmu je princip stejný jako u předchozího algoritmu. Rozdíl je pouze v tom, že vybraní jedinci jsou odstraňovány z populace, ze které lze vybírat (řádek 38). Zároveň musí být odstraněna z ruletového kola jejich výseč (řádek 39). To znamená, že musí být přepočteny horní a dolní meze výsečí všech následujících jedinců (řádek 32, 33). Pokud padl výběr na prvního jedince jeho spodní mez je nula (řádek 31).

$X_{MP} \longleftarrow$ RouletteWheelSelect$_w(X_{Pop}, m_{MP})$		
Funkce, která na základě ruletového mechanismu vybírá m_{MP} jedinců ze seznamu X_{Pop}. Výstupem funkce je seznam jedinců určených pro reprodukci. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	i:	Proměnná - čítač
Data:	a, b:	Dočasné proměnné pro uložení numerických hodnot
Data:	A:	Pole hodnot fitness
Data:	$minf, maxf, sum$:	Hodnoty minima, maxima a součtu hodnot fitness
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     A ⟵ CreateList(Length(X_Pop), 0);
3     minf ⟵ ∞;
4     maxf ⟵ −∞;
5     for i ⟵ 0 to Length(X_Pop) − 1 do begin
6         a ⟵ f(X_Pop[i]);
7         A[i] ⟵ a
8         if a < minf then minf ⟵ a;
9         if a > maxf then maxf ⟵ a;
10    end;
11    if maxf = minf then begin
12        maxf ⟵ maxf + 1;
13        minf ⟵ minf − 1;
```

```
14      end;
15      sumNormf ← 0;
16      for i ← 0 to Length(X_Pop) − 1 do begin
17          Normf ← (maxf − A[i]) / (maxf − minf);
18          A[i] ← Normf;
19          sumNormf ← sumNormf + Normf;
20      end;
21      sum ← 0;
22      for i ← 0 to Length(X_Pop) − 1 do begin
23          Pr ← A[i] / sumNormf;
        //pravděpodobnost výběru jedince
24          sum ← sum + Pr;
        //kumulativní ohodnocení
25          A[i] ← sum;
        //vektor kumulativních ohodnocení
26      end;
27      for j ← 0 to min{m_MP, Length(X_Pop)} − 1 do begin
        //min je v tomto případě funkce, která navrací nejmenší hodnotu z množiny
28          a ← Random_u(0, sum);
29          for i ← 0 to Length(X_Pop) − 1 do
30              if (A[i] > a) then break;
        //nalezení segmentu v ruletě tj. i-tého indexu jedince v populaci
31          if i = 0 then b ← 0
32          else b ← A[i − 1];
33          b ← A[i] − b;
34          for k ← i + 1 to Length(A) − 1 do
35              A[k] ← A[k] − b;
36          sum ← sum − b;
37          X_MP ← AddListItem(X_MP, X_Pop[i]);
38          X_Pop ← DeleteListItem(X_Pop, i);
39          A ← DeleteListItem(A, i);
40      end;
41      Result ← X_MP;
42  end;
```

Algoritmus 3-10 RouletteWheelSelect$_w$

3.2.4 Turnajová selekce (*Tournament Selection*)

- TournamentSelect$_r$:

V následujícím algoritmu (viz Algoritmus 3-11 TournamentSelect$_r$) je popsán proces turnajové selekce se substitucí. Nejprve je vytvořena prázdná množina jedinců, do které budou postupně umístěni vybraní jedinci (řádek 2). Jedinci jsou setříděni na základě svých fitness hodnot (řádek 3). Do populace pro páření bude vybráno tolik jedinců, kolik je hodnota m_{MP} (řádek 4). Náhodně vybraný jedinec – značí ho index ve skupině (řádek 5), bude zápasit s dalšími (řádek 6) náhodně vybranými jedinci – značí je též index ve skupině (řádek 7) v *k*-1 párových zápasech. Zápas vyhraje a bude vybrán do populace k páření ten jedinec (řádek 8), který má nejlepší hodnotu fitness ze všech vybraných jedinců v *k*-1 soutěžních kolech (v případě minimalizace - menší hodnotu fitness – viz řádek 7). Jako výsledek tohoto algoritmu je navrácena populace obsahující jedince k páření (řádek 10). V algoritmu může jedinec soupeřit sám proti sobě – své kopii. Tato situace je svým

způsobem v realitě neproveditelná, ale uvědomme si, že takoví jedinci mají stejnou hodnotu fitness a v takovém případě vyhraje první jedinec a ne jeho kopie, protože díky párovému porovnání v algoritmu (řádek 7) je vybrán první, svým způsobem pravý, jedinec.

$X_{\mathrm{MP}} \longleftarrow \mathrm{TournamentSelect}_r(X_{\mathrm{Pop}}, m_{\mathrm{MP}}, k)$		
Funkce, která na základě k zápasů mezi náhodně vybranými jedinci vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	k:	Počet jedinců, mezi nimiž se uspořádá turnaj
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	a:	Index jedince, který vyhrál zápas
Data:	i:	Proměnné - čítače
Data:	j:	Počet uskutečněných zápasů vybraného jedince proti soupeřům
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     X_MP ← ( );
3     X_Pop ← Sort_a(X_Pop, CF_f(x));
4     for i ← 0 to m_MP − 1 do begin
5       a ← ⌊Random_u(0,Length(X_Pop))⌋;
6       for j ← 1 to k − 1 do
7         a ← min{a, ⌊Random_u(0,Length(X_Pop))⌋};
8       X_MP ← AddListItem(X_MP, X_Pop[a]);
9     end;
10    Result ← X_MP;
11  end;
```

Algoritmus 3-11 TournamentSelect$_r$ **[15]**

- TournamentSelect$_{w1}$:

Turnajová selekce bez substituce (viz Algoritmus 3-12) je, v podstatě taková varianta algoritmu, každý vítěz zápasu, už se nemusí účastnit dalších zápasů – je odstraněn z vstupní populace (řádek 9) – listiny, kde jsou zapsáni všichni jedinci, kteří se mohou zápasit.

$X_{\mathrm{MP}} \longleftarrow \mathrm{TournamentSelect}_{w1}(X_{\mathrm{Pop}}, m_{\mathrm{MP}}, k)$		
Funkce, která na základě k zápasů mezi náhodně vybranými jedinci vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	k:	Počet jedinců, mezi nimiž se uspořádá turnaj
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	a:	Index jedince, který vyhrál zápas
Data:	i:	Proměnné - čítače
Data:	j:	Počet uskutečněných zápasů vybraného jedince proti soupeřům
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     X_MP ← ( );
3     X_Pop ← Sort_a(X_Pop, CF_f(x));
4     for i ← 0 to min{Length(X_Pop), m_MP} − 1 do begin
5       a ← ⌊Random_u(0,Length(X_Pop))⌋;
6       for j ← 1 to min{Length(X_Pop), k} − 1 do
7         a ← min{a, ⌊Random_u(0,Length(X_Pop))⌋};
8       X_MP ← AddListItem(X_MP, X_Pop[a]);
9       X_Pop ← DeleteListItem(X_Pop, a);
10    end;
11    Result ← X_MP;
12  end;
```

Algoritmus 3-12 TournamentSelect$_{w1}$ **[15]**

- TournamentSelect$_{w2}$ – podstata tohoto algoritmu je totožná s předchozím algoritmem, s tím rozdílem, že všichni jedinci – jejich indexy - jsou vybíráni do jednoho celkového kola – jednoho seznamu (řádek 10), ve kterém už se jedinec nesmí nacházet (řádek 9). Z tohoto seznamu je vybrán hlavní vítěz s nejmenší hodnotou fitness (řádek 12). Tento postup je opakován tolikrát, kolik má být vybráno jedinců k páření (řádek 4).

$X_{MP} \leftarrow$ TournamentSelect$_{w2}(X_{Pop}, m_{MP}, k)$		
Funkce, která na základě k zápasů mezi náhodně vybranými jedinci vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	k:	Počet jedinců, mezi nimiž se uspořádá turnaj
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	A:	Seznam soupeřů v zápasu
Data:	a:	Index jedince, který vyhrál zápas
Data:	i:	Proměnné - čítače
Data:	j:	Počet uskutečněných zápasů vybraného jedince proti soupeřům
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     X_MP ← ( );
3     X_Pop ← Sort_a(X_Pop, CF_f(x));
4     for i ← 0 to m_MP − 1 do begin
5       A ← ( );
6       for j ← 1 to min{k,Length(X_Pop)} do begin
7         repeat
8           a ← ⌊Random_u(0,Length(X_Pop))⌋;
9         until Search_u(a, A) < 0;
10        A ← AddListItem(A, a);
11      end;
12        a ← min{A};
```

```
13          X_MP ⟵ AddListItem(X_MP, X_Pop[a]);
14        end;
15        Result ⟵ X_MP;
16      end;
```

Algoritmus 3-13 TournamentSelect$_{w2}$ **[15]**

- TournamentSelect$_r^p$ – algoritmus odráží předchozí chování turnajové selekce se substitucí je (viz Algoritmus 3-14). Je dobré si uvědomit, že tento algoritmus je v podstatě shodný s algoritmem TournamentSelect$_r$ (viz Algoritmus 3-14) pokud by byla nastavena pravděpodobnost $p = 1$.

$X_{\mathrm{MP}} \longleftarrow \mathrm{TournamentSelect}_{r,p}(X_{\mathrm{Pop}}, m_{\mathrm{MP}}, k, p)$		
Funkce, která na základě k zápasů mezi náhodně vybranými jedinci vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Nejlepší jedinec zápasu je vybrán s pravděpodobností p. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	k:	Počet jedinců, mezi nimiž se uspořádá turnaj
Vstup:	p:	Pravděpodobnost výběru, $p \in [0,1]$
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	A:	Seznam soupeřů v zápasu
Data:	a:	Index jedince, který vyhrál zápas
Data:	i:	Proměnné - čítače
Data:	j:	Počet uskutečněných zápasů vybraného jedince proti soupeřům
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1       begin
2         X_MP ⟵ ( );
3         X_Pop ⟵ Sort_a(X_Pop, CF_f(x));
4         for i ⟵ 0 to m_MP − 1 do begin
5           A ⟵ ( );
6           for j ⟵ 0 to k − 1 do
7             A ⟵ AddListItem(A, ⌊Random_u(0,Length(X_Pop))⌋);
8           A ⟵ Sort_a (A,CF(a_1,a_2)≡(f(a_1) − f(a_2)));
9           for j ⟵ 0 to Length(A) − 1 do
10            if (Random_u( ) ≤ p) ∨ (j ≥ Length(A) − 1) then begin
11              X_MP ⟵ AddListItem(X_MP, X_Pop[A[j]]);
12              j ⟵ ∞;
13            end;
14        end;
15        Result ⟵ X_MP;
16      end;
```

Algoritmus 3-14 TournamentSelect$_{r,p}$ **[15]**

3.2.5 Selekce na základě uspořádání (*Ordered Selection*)

❖ $\mathrm{OrderedSelect}_r$:

Selekce na základě uspořádání – se substitucí – v algoritmu je nejprve vypočtena mocnina indexu (řádek 2). Následuje vytvoření prázdné množiny, do které budou vkládání postupně vybraní jedinci (řádek 3). Celá vstupní populace je seřazena na základě hodnot fitness jejích jedinců (řádek 4). Náhodně vybraný index jedince je vygenerován na základě rovnoměrného rozdělení (řádek 6) a poté je umocněn již zmiňovanou vypočtenou mocninou (řádek 6). Takto nově přepočtený index jedince je vynásobený délkou populace a oříznut na nejmenší celé číslo (řádek 6) aby vyhovoval podmínce, že indexy jedinců v populaci mohou nabývat hodnot od 0 do velikost populace-1. Následně je pak vybrán jedinec s tímto indexem, který je zařazen do populace pro páření (řádek 6). Takové kroky jsou prováděny tak dlouho, dokud není dosaženo definovaného počtu jedinců v populaci pro páření (řádek 5). Výsledkem poskytovaným tímto algoritmem je populace určená k páření.

$X_{\mathrm{MP}} \longleftarrow \mathrm{OrderedSelect}_r(X_{\mathrm{Pop}}, m_{\mathrm{MP}}, k)$	
Funkce, která na základě pravděpodobnosti výběru jedince, jenž je úměrná jeho pozici v setříděném seznamu jedinců populace vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce se substitucí – stejný jedinec se může vyskytnout ve výstupním seznamu několikrát.	
Vstup:	X_{Pop}: Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}: Počet jedinců, kteří budou umístěni do výsledné populace určené k páření
Vstup:	k: Selekční tlak - počet očekávaných potomků nejlepšího jedince, $k \neq m_{MP}$
Data:	f: Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	q: Mocnina indexu náhodně vygenerovaného jedince podle rovnoměrného rozdělení užitá pro seřazení
Data:	i: Proměnná - čítač
Výstup:	X_{MP}: Výsledná populace určená k páření

1 **begin**
2 $q \longleftarrow \frac{1}{1-\frac{\log k}{\log m_{MP}}}$;
3 $X_{\mathrm{MP}} \longleftarrow ( )$;
4 $X_{\mathrm{Pop}} \longleftarrow \mathrm{Sort}_{\mathrm{a}}(X_{\mathrm{Pop}}, \mathrm{CF}_{\mathrm{f(x)}})$;
5 **for** $i \longleftarrow 0$ **to** $m_{\mathrm{MP}} - 1$ **do**
6 $X_{\mathrm{MP}} \longleftarrow \mathrm{AddListItem}(X_{\mathrm{MP}}, X_{\mathrm{Pop}}[\lfloor \mathrm{Random}_{\mathrm{u}}( )^{q} * \mathrm{Length}(X_{\mathrm{Pop}}) \rfloor])$;
7 **Result** $\longleftarrow X_{\mathrm{MP}}$;
8 **end**;

Algoritmus 3-15 $\mathrm{OrderedSelect}_r$ **[15]**

❖ $\mathrm{OrderedSelect}_w$:

Selekce na základě uspořádání – bez substituce – následující algoritmus má shodnou podstatu s předcházejícím algoritmem se substitucí. Rozdíl je pouze v tom, že jedinec – jeho index, který byl vybrán do populace k páření (řádek 6), je následně odstraněn z populace X_{Pop} (řádek 8), ze které se vybírají jedinci do populace pro páření X_{MP} (řádek 7). Do populace určené k páření lze maximálně vybrat tolik jedinců, kolik je minimální hodnota z definovaného počtu jedinců nebo velikost vstupní populace (řádek 5).

$X_{\mathrm{MP}} \longleftarrow \mathrm{OrderedSelect}_w(X_{\mathrm{Pop}}, m_{\mathrm{MP}}, k)$		
Funkce, která na základě pravděpodobnosti výběru jedince, jenž je úměrná jeho pozici v setříděném seznamu jedinců populace vybírá m_{MP} vítězných jedinců ze seznamu X_{Pop} do výsledného seznamu jedinců X_{MP} určených k další reprodukci – tento seznam je výstupem funkce. Selekce bez substituce – jedinec se může vyskytnout ve výstupním seznamu maximálně jedenkrát.		
Vstup:	X_{Pop}:	Seznam, ze kterého bude proveden výběr do X_{MP}
Vstup:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Vstup:	k:	Selekční tlak - počet očekávaných potomků nejlepšího jedince, $k \neq m_{MP}$
Data:	f:	Funkce, která poskytuje jako výstup hodnotu fitness jedince
Data:	q:	Mocnina indexu náhodně vygenerovaného jedince podle rovnoměrného rozdělení užitá pro seřazení
Data:	i:	Proměnná - čítač
Výstup:	X_{MP}:	Výsledná populace určená k páření

```
1   begin
2     q ⟵ 1 / (1 − log k / log m_MP);
3     X_MP ⟵ ( );
4     X_Pop ⟵ Sort_a(X_Pop, CF_f(X));
5     for i ⟵ 0 to min{Length(X_Pop), m_MP} − 1 do begin
6       j ⟵ ⌊Random_u( )^q * Length(X_Pop)⌋;
7       X_MP ⟵ AddListItem(X_MP, X_Pop[j]);
8       X_Pop ⟵ DeleteListItem(X_Pop, j);
9     end;
10    Result ⟵ X_MP;
11  end;
```

Algoritmus 3-16 $\mathrm{OrderedSelect}_w$ **[15]**

4 Vývoj aplikace simulační optimalizace

4.1 Implementované optimalizační metody

V kapitole budou uvedeny modifikované optimalizační metody implementované ve formě pseudopascalovských algoritmů. Tyto algoritmy byly implementovány v aplikaci simulační optimalizace.

4.1.1 Náhodné prohledávání - Random Search

$\mathbf{X}^* \longleftarrow \mathrm{RandomSearch}(A, B)$		
Funkce, jejímž výstupem je dosud nalezený nejlepší prvek. Generování souřadnic rozhodovacích proměnných prvku se děje na základě rovnoměrného rozdělení v celém prohledávaném prostoru.		
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	$\mathbf{X}$:	Nově vygenerovaný prvek pomocí mutace
Data:	$F(\mathbf{X})$:	Hodnota účelové funkce prvku $\mathbf{X}$
Data:	i:	Proměnná - čítač
Výstup:	$\mathbf{X}^*$:	Dosud nalezený nejlepší prvek

```
1   begin
```

```
2       X* ⟵ Create(A, B); //počáteční přípustné řešení
3       while not TerminationCriterion( ) do begin      //kritérium ukončení
4           X ⟵ Create(A, B);  //nově vygenerovaný prvek
5           if F(X) < F(X*) then
6               X* ⟵ X;  //bylo nalezeno lepší řešení než aktuální – nahrazení
7       end;
8       result ⟵ X*;
9     end;
```

Algoritmus 4-1 RandomSearch

- TerminationCriterion – specifikované kritérium ukončení např.
  - Dosažení specifikované hodnoty - VTR – Value To Reach.
  - Překročení výpočetního času.
  - Dosažení specifikovaného celkového počtu iterací.
  - Nejsou prováděna žádná zlepšení prvků atd.

X ⟵ Create (A, B)		
Funkce, jejímž výstupem je náhodně vygenerovaný prvek na základě rovnoměrného rozdělení v mezích intervalu prohledávaného prostoru.		
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	n:	Dimenze prostoru všech rozhodovacích proměnných
Data:	j:	Čítač
Výstup:	**X**:	Vygenerovaný prvek – seznam rozhodovacích proměnných

```
1     begin
2       X ⟵ ( );
3       n ⟵ min(Length(A), Length(B));
      (*zjištění počtu rozhodovacích proměnných – délky seznamu A i B by měly být stejné*)
4       for j ⟵ 0 to n − 1 do
5           X ⟵ AddListItem(X, Random_u(A[j], B[j]));
      (*vygenerování hodnoty rozhodovací proměnné, která bude vložena do prvku*)
6       result ⟵ X;
7     end;
```

Algoritmus 4-2 Create **- generování prvku**

- AddListItem - viz kapitola 1, str. 8.
- $\min(\text{Length}(A), \text{Length}(B))$ – funkce, která vybírá menší hodnotu ze dvou čísel v závorce
- Length – viz kapitola 1, str. 8.

Tento a následující optimalizační algoritmy mohou také využívat operace pro množinu optim, namísto zachování jediného nejlepšího prvku.

X* ⟵ RandomSearch(A, B)		
Funkce, jejímž výstupem jsou vybrané nalezené kvalitní prvky namísto jediného řešení. Generování souřadnic rozhodovacích proměnných prvku se děje na základě rovnoměrného rozdělení v celém prohledávaném prostoru.		
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných

Data:	$\mathbf{X}$:	Nově vygenerovaný prvek pomocí mutace
Data:	$F(\mathbf{X})$:	Hodnota účelové funkce prvku $\mathbf{X}$
Data:	i:	Proměnná - čítač
Výstup:	$\mathbf{X}^*$:	Dosud nalezený nejlepší prvek

```
1   begin
2     X* ⟵ Create(A, B); //počáteční přípustné řešení je zároveň dosud nejlepším řešení
3     X* ⟵ UpdateOptimalSet(X*, X*);        //přidání prvku do množiny optim
4     X* ⟵ PruneOptimalSet(X*, X*);        //oříznutí množiny optim
5     while not TerminationCriterion( ) do begin       //kritérium ukončení
6         X ⟵ Create(A, B);  //nově vygenerovaný prvek
7         X* ⟵ UpdateOptimalSet(X*, X);       //přidání prvku do množiny optim
8         X* ⟵ PruneOptimalSet(X*, X);        //oříznutí množiny optim
9         if F(X) < F(X*) then    //bylo nalezeno lepší řešení než aktuální
10            X* ⟵ X;  //bylo nalezeno lepší řešení než aktuální – nahrazení
11    end;
12    result ⟵ ExtractOptimalSet(X*);      //poskytnutí výsledků z množiny optim
13  end;
```

Algoritmus 4-3 RandomSearch s využitím množiny optim

$X^*_{\text{New}} \longleftarrow \text{UpdateOptimalSet}(X^*_{\text{Old}}, \mathbf{X}^*)$		
Vytvoření nové prázdné množiny a postupné vložení optimálních prvků		
Vstup:	X^*_{Old}:	Množina - seznam optim známá před vygenerováním $\mathbf{X}^*$
Vstup:	$\mathbf{X}^*$:	Nový prvek, který bude testován
Výstup:	X^*_{New}:	Množina - seznam optim aktualizovaná pomocí prvku $\mathbf{X}^*$

```
1   begin
2     X*New ⟵ ( );
3     for i ⟵ 0 to Length(X*Old ) − 1 do
4         if CF_F(X)(X*, X*Old[i]) > 0 then          begin
      (*nový prvek X* je horší než původní prvek X*Old[i], pak se do nové aktualizované množiny přidá
      prvek z původní množiny optim)
5             X*New ⟵ AddListItem(X*New, X*Old[i]);
6             if CF_F(X)(X*, X*Old[i]) ≥ 0 then begin
      (*nový prvek X* je stejný nebo horší než původní prvek X*Old[i], pak je jako výsledek funkce
      navrácena původní množina optim*)
7                 result ⟵ X*Old;
8                 exit;   //opuštění funkce
9             end;
10        end;
11    X*New ⟵ AddListItem(X*New, X*);
      (*vložení nového prvku na konec seznamu nové aktualizované množiny optim*)
12    result ⟵ X*New;
13  end;
```

Algoritmus 4-4 UpdateOptimalSet **[10]**

$l \longleftarrow \mathrm{CF}_{F(\mathbf{X})}(\mathbf{X}_1, \mathbf{X}_2)$		
Funkce, jejímž výstupem je (-1) v případě, že první prvek je lepší než druhý – v případě minimalizace účelové funkce, hodnota účelové funkce prvního prvku je menší než hodnota účelové funkce druhého prvku. Výstup (1) nastává v případě, že první prvek je horší než druhý, a výstupem je (0) pokud jsou si prvky rovny.		
Vstup:	$\mathbf{X}_1$:	První porovnávaný prvek
Vstup:	$\mathbf{X}_2$:	Druhý porovnávaný prvek
Data:	$F(\mathbf{X})$:	Účelová funkce
Výstup:	l:	Výstup komparační funkce – možné výstupy $\{-1,1,0\}$

```
6   begin
7     if F(X1) < F(X2) then result ⟵ −1      //první prvek je lepší než druhý
8     else  if F(X1) > F(X2) then result ⟵ 1      //první prvek je horší než druhý
9           else result ⟵ 0; //prvky jsou stejné
10  end;
```

Algoritmus 4-5 $\mathrm{CF}_{F(\mathbf{X})}$ - Komparace hodnot účelové funkce dvou prvků v případě minimalizace účelové funkce

$X^* \longleftarrow \mathrm{ExtractOptimalSet}(X_{\mathrm{Any}})$		
Extrakce množiny optim X^* z dané množiny X_{Any}		
Vstup:	X_{Any}:	Množina, ze které jsou extrahovány prvky množiny optim X^*
Data:	i, j:	Proměnné - čítače
Výstup:	X^*:	Množina optim extrahovaná z množiny X_{Any}

```
1   begin
2     X* ⟵ X_Any;
3     i ⟵ Length(X*) − 1;
4     while i > 0 do begin
5       j ⟵ i − 1;
6       while j ≥ 0 do begin
7         if CF_F(X)(X*[i], X*[j]) < 0 then begin
        //prvek s indexem i je lepší než prvek s indexem j
8           X* ⟵ DeleteListItem(X*, j);
9           i ⟵ i − 1;
10        end
11        else
12          if CF_F(X)(X*[i], X*[j]) > 0 then begin
        //prvek s indexem i je horší než prvek s indexem j
13            X* ⟵ DeleteListItem(X*, i);
14            j ⟵ −1;
15          end;
16        j ⟵ j − 1;
17      end;
18      i ⟵ i − 1;
19    end;
20    result ⟵ X*;
21  end;
```

Algoritmus 4-6 ExtractOptimalSet [10]

- ❖ PruneOptimalSet - operace, která ořezává množinu dosavadních optim získanou během optimalizace na novou množinu o velikosti m^*. Tato operace může být založena např. na algoritmu seskupování (*Clustering Algorithm*). Princip spočívá v rozložení prohledávaného prostoru na disjunktní množiny prvků – clustery. Pro takové clustery se vypočtou jejich jádra. Pomocí nich lze např. zamítat body, které jsou dále než specifikovaná vzdálenost od jádra. Při ořezávání lze namísto algoritmu seskupování využít i jiných postupů.

4.1.2 Downhill Simplex

$\mathbf{X}^* \longleftarrow \mathrm{DownhillSimplex}(F, n, \alpha, \gamma, \rho, \sigma, A, B)$		
Algoritmus, který pracuje s množinou bodů označovanou jako „Simplex“. Tato množina bodů, obsahuje minimálně $n+1$ bodů, kde n vyjadřuje rozměr prostoru rozhodovacích proměnných. Tyto body jsou nekomplanární (lineárně nezávislé body). Základními fázemi algoritmu jsou reflexe, expanze, kontrakce a redukce.		
Vstup:	F:	Účelová funkce
Vstup:	n:	Dimenze prohledávaného prostoru $\tilde{X}$
Vstup:	α: γ: ρ: σ:	koeficient reflexe (obvykle $\alpha = 1$) koeficient expanze (obvykle $\gamma = 1$) koeficient kontrakce (obvykle $\rho = \frac{1}{2}$) koeficient redukce (obvykle $\sigma = \frac{1}{2}$)
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	S:	Simplex – množina nekomplanárních $n+1$ bodů
Data:	$\mathbf{M}$:	Těžiště simplexu
Data:	$\mathbf{R}$:	Bod získaný reflexí
Data:	$\mathbf{E}$:	Bod získaný expanzí
Data:	$\mathbf{C}$:	Bod získaný kontrakcí
Data:	$\mathbf{E}$:	Bod získaný expanzí
Data:	i:	Proměnná – čítač
Data:	b:	Logická proměnná – True, False
Výstup:	X^*:	Globální optimum funkce (kandidát řešení)

```
1   begin
2     S ⟵ CreatePop(n + 1, A, B);
3     while not TerminationCriterion( ) do begin
4         S ⟵ Sort_a(S, CF_F(X));
    //sestupné setřídění S podle F(X), v případě maximalizace účel. funkce sestupné třídění
5         M ⟵ (1/n) Σ_{i=0}^{n-1} S[i];    //těžiště všech prvků simplexu kromě nejhoršího prvku
6         R ⟵ M + α(M − S[n]);
    //reflexe: zrcadlení nejhoršího prvku přes těžiště simplexu M; S[n] nejhorší kandidát řešení
7         if F(S[0]) < F(R) < F(S[n]) then S[n] ⟵ R //minimalizace účelové funkce
8         else    if F(R) < F(S[0]) then begin
9                     E ⟵ R + γ(R − M);
    //expanze: hledání odlehlejších prvků v tomto směru
10                    if F(E) < F(R) then S[n] ⟵ E
11                    else S[n] ⟵ R;
12                end
13                else begin
```

```
14                  b ⟵ True;
15                  if F(R) ≥ F(S[n − 1]) then begin
    //kontrakce: testovací prvek mezi R a M
16                      C ⟵ ρR + (1 − ρ)M;
17                      if F(C) ≤ F(R) then begin
18                          S[n] ⟵ C;
19                          b ⟵ False;
20                      end;
21                  end;
22                  if b then
    //redukce: smrštění směrem k nejlepšímu prvku S[0]
23                      for i ⟵ n downto 1 do
24                          S[i] ⟵ S[0] + σ(S[i] − S[0]);
25              end;
26    Result⟵ S[0];
27    end;
28  end;
```

Algoritmus 4-7 DownhillSimplex [15]

- $\mathrm{Sort_a}$ - viz kapitola 1, str. 8.

$X_{\mathrm{Pop}} \longleftarrow \mathrm{CreatePop}\ (m, A, B)$		
Funkce, jejímž výstupem jsou náhodně vygenerované prvky na základě rovnoměrného rozdělení v mezích intervalu prohledávaného prostoru.		
Vstup:	m:	Počet prvků v seznamu
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	n:	Dimenze prostoru všech rozhodovacích proměnných
Data:	i:	Čítač
Výstup:	X_{Pop}:	Vygenerovaný seznam prvků

```
1   begin
2     X_Pop ⟵ ( );
3     for i ⟵ 0 to m − 1 do
4         X_Pop ⟵ AddListItem (X_Pop, Create (A, B));
    (*postupné plnění seznamu prvky*)
5     result ⟵ X_Pop;
6   end;
```

Algoritmus 4-8 CreatePop - generování množiny prvků

4.1.3 Stochastický horolezecký algoritmus - Hill Climbing

$\mathbf{X}^* \longleftarrow \mathrm{StochasticHillClimbing}(m, E, A, B)$	
Funkce, jejímž výstupem je dosud nalezený nejlepší prvek. Generování prvků je prováděno v sousedství nejlepšího nalezeného prvku z předchozích m prvků (populace). Generování je prováděno pomocí transformace (mutace) nejlepšího prvku z X_{Pop} na základě rovnoměrného rozdělení.	
Vstup:	m: Velikost seznamu (populace) X_{Pop} tj. délka seznamu $m = \mathrm{Length}(X_{\mathrm{Pop}})$
Vstup:	E: Seznam obsahující délky intervalů pro jednotlivé rozhodovací proměnné – velikosti stran sousedství prvku
Vstup:	A: Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B: Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	$\mathbf{X}$: Nově vygenerovaný prvek pomocí mutace
Data:	X_{Pop}: Seznam vygenerovaných prvků (populace)
Data:	$\mathbf{X}_{\mathrm{Best}}$: Nejlepší prvek (jedinec) z předchozí skupiny prvků (jedinců z předchozí populace)
Data:	$F(\mathbf{X})$: Hodnota účelové funkce prvku $\mathbf{X}$
Data:	i: Proměnná - čítač
Výstup:	$\mathbf{X}^*$: Dosud nalezený nejlepší prvek

```
1   begin
2     X_Pop ← CreatePop(m, A, B);    //počáteční skupina přípustných řešení
3     X_Pop ← Sort_a(X_Pop, CF_F(X));   //setřídění X_Pop podle hodnoty účelové funkce
4     X_Best ← X_Pop[0];    //nejlepší prvek z X_Pop
5     X* ← X_Best;      //nejlepší prvek z X_Pop je zároveň nejlepší dosud nalezený prvek
6     while not TerminationCriterion( ) do begin      //kritérium ukončení
7       X_Pop ← ( );  //vyčištění X_Pop
8       for i ← 0 to m − 1 do begin
9         X ← Mutate_u(X_Best, E);
      (*mutace nejlepšího prvku z X_Pop na základě rovnoměrného rozdělení*)
10        X ← Perturbation_u(X, A, B);      //perturbace prvku
11        X_Pop ← AddListItem(X_Pop, X);    //přidání vygenerovaného prvku do X_Pop
12      end;
13      X_Pop ← Sort_a(X_Pop, CF_F(X));
14      X_Best ← X_Pop[0];
15      if F(X_Best) < F(X*) then   //bylo nalezeno lepší řešení než aktuální
16        X* ← X_Best ;
17    end;
18    result ← X*;
19  end;
```

Algoritmus 4-9 StochasticHillClimbing

- CreatePop – viz Algoritmus 4-8, str. 28.
- $\mathrm{Sort_a}$ - viz kapitola 1, str. 8.
- TerminationCriterion – viz kapitola 4.1.1, str. 23.
- Perturbation – viz Algoritmus 4-11, str. 30.
- AddListItem – viz kapitola 1, str. 8.

$\mathbf{X}_{\mathrm{Mut}} \longleftarrow \mathrm{Mutate}_{\mathrm{u}}(\mathbf{X}, E)$	
Funkce, jejímž výstupem je zmutovaný prvek (vektor představuje seznam hodnot rozhodovacích proměnných), který vznikl transformací rozhodovacích proměnných (v kontextu evolučních algoritmů mutací) vstupního prvku $\mathbf{X}$. Transformace jednotlivých složek vstupního prvku se provádí na základě vygenerování náhodné čísla generovaného podle rovnoměrného rozdělení v rozmezí délky intervalu dolní (včetně této meze) a horní meze (není prvkem tohoto intervalu) definovaného pro každou jednotlivou rozhodovací proměnnou (geometricky pro každou osu).	
Vstup:	$\mathbf{X}$: Původní prvek, jehož souřadnice budou transformovány – seznam, jehož prvky budou transformovány
Vstup:	E: Seznam obsahující jednotlivé délky intervalů (vzdálenost mezi dolní a horní mezí) pro jednotlivé rozhodovací proměnné užité pro generování náhodného čísla ($E[j] = \left\|\tilde{X}_j\right\| = \left\|b_j - a_j\right\|$)
Data:	j: Proměnná - čítač
Výstup:	$\mathbf{X}_{\mathrm{Mut}}$: Prvek - seznam, jehož složky jsou zmutované - transformované

```
1   begin
2     Randomize;
3     X_Mut ⟵ ( );
4     for j ⟵ 0 to (Length(X) − 1) do        //pro všechny rozhodovací proměnné
5         X_Mut ⟵ AddListItem(X_Mut, Random_u(X[j] − |E[j]|/2, X[j] + |E[j]|/2));
      (*přidání hodnoty souřadnice do vektoru X_Mut pomocí funkce pro generování náhodného čísla v
      mezích intervalu podle rovnoměrného rozdělení*)
6     result ⟵ X_Mut;
7   end;
```

Algoritmus 4-10 $\mathbf{Mutate}_{u}$ - Transformace složek prvku na základě rovnoměrného rozdělení

$\mathbf{X}_{\mathrm{Pert}} \longleftarrow \mathrm{Perturbation}(\mathbf{X}, A, B)$	
Funkce, jejímž výstupem je prvek, který vznikl perturbací – zrcadlením - souřadnic rozhodovacích proměnných vstupního prvku, které ležely mimo rozsah intervalu jednotlivé rozhodovací proměnné.	
Vstup:	$\mathbf{X}$: Prvek, který obsahuje souřadnice rozhodovacích proměnných mimo prohledávaný interval rozhodovací proměnné
Vstup:	A: Seznam obsahující dolní meze intervalů pro jednotlivé rozhodovací proměnné
Vstup:	B: Seznam obsahující horní meze intervalů pro jednotlivé rozhodovací proměnné
Data:	j: Čítač – vyjadřuje index rozhodovací proměnné
Výstup:	$\mathbf{X}_{\mathrm{Pert}}$: Prvek vzniklý perturbací původního prvku.

```
1   begin
2     X_Pert ⟵ X;
3     for j ⟵ 0 to Length(X) − 1 do
      //indexování rozhodovacích proměnných - dimenzí
4       while (X[j] < A[j]) or (X[j] > B[j]) do
5         if (X[j] < A[j]) then
6           X_Pert[j] ⟵ 2 · A[j] − X[j]
7         else if (X[j] > B[j]) then
8                 X_Pert[j] ⟵ 2 · B[j] − X[j];
9     result ⟵ X_Pert[j];
10  end;
```

Algoritmus 4-11 $\mathbf{Perturbation}$ - Perturbace souřadnic prvku mimo meze [14]

4.1.4 **Stochastické zakázané prohledávání - Tabu Search**

$\mathbf{X}^* \leftarrow \text{StochasticTabuSearch}(m, m_{\text{Tabu}}, E, A, B)$		
Funkce, jejímž výstupem je dosud nalezený nejlepší prvek. Během optimalizačního procesu je každý nově vygenerovaný prvek vložen do seznamu zakázaných prvků. Takový prvek nesmí být navštíven, pokud nesplňuje aspirační kritérium. Pokud je překročena povolená délka seznamu zakázaných prvků, je provedeno vyjmutí prvku z tohoto seznamu – metoda FIFO. Generování prvku v sousedství dosud nejlepšího nalezeného prvku z předchozích m prvků (populace), se děje pomocí transformace (mutace) nejlepšího prvku z X_{Pop} na základě rovnoměrného rozdělení.		
Vstup:	m:	Velikost seznamu (populace) X_{Pop} tj. délka seznamu $m = \text{Length}(X_{\text{Pop}})$
Vstup:	m_{Tabu}:	Maximální délka seznamu obsahující prvky vytvořené v m_{Tabu} předchozích krocích
Vstup:	E:	Seznam obsahující délky intervalů pro jednotlivé rozhodovací proměnné – velikosti stran sousedství prvku
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	X_{Pop}:	Seznam vygenerovaných prvků (populace)
Data:	$\mathbf{X}$:	Nově vygenerovaný prvek pomocí mutace
Data:	$F(\mathbf{X})$:	Hodnota účelové funkce prvku $\mathbf{X}$
Data:	$\mathbf{X}_{\text{Best}}$:	Nejlepší prvek (jedinec) z předchozí skupiny prvků (jedinců z předchozí populace)
Data:	X_{Tabu}:	Seznam obsahující zakázané prvky, které nebudou akceptovány
Data:	i:	Proměnná - čítač
Výstup:	$\mathbf{X}^*$:	Dosud nalezený nejlepší prvek

```
1   begin
2     X_Pop ← CreatePop(m, A, B);    //počáteční skupina přípustných řešení
3     X_Pop ← Sort_a(X_Pop, CF_F(X));   //setřídění X_Pop podle hodnoty účelové funkce
4     X_Best ← X_Pop[0];    //nejlepší prvek z X_Pop
5     X* ← X_Best;      //nejlepší prvek z X_Pop je zároveň nejlepší dosud nalezený prvek
6     X_Tabu ← ( ); //vytvoření prázdného seznamu obsahujícího zakázané prvky
7     X_Tabu ← AppendList(X_Tabu, X_Pop);
8     while not TerminationCriterion( ) do begin      //kritérium ukončení
9       X_Pop ← ( );  //vyčištění X_Pop
10      for i ← 0 to m − 1 do begin
11        repeat
12          X ← Mutate_u(X_Best, E);
      (*mutace nejlepšího prvku z X_Pop na základě rovnoměrného rozdělení*)
13          X ← Perturbation_u(X, A, B);  //perturbace prvku
14          X_Pop ← AddListItem(X_Pop, X);      //přidání vygenerovaného prvku do X_Pop
15        until (Search_u(X, X_Tabu) < 0) or (F(X) < F(X*));
      (*prvek se nenachází v X_Tabu nebo splňuje aspirační kritérium*)
16        if Length(X_Tabu) ≥ m_Tabu then
17          X_Tabu ← DeleteListItem(X_Tabu, 0);
18        X_Tabu ← AddListItem(X_Tabu, X);
19        X_Pop ← AddListItem(X_Pop, X);    //přidání vygenerovaného prvku do X_Pop
20      end;
21      X_Pop ← Sort_a(X_Pop, CF_F(X));
22      X_Best ← X_Pop[0];
23      if F(X_Best) < F(X*) then   //bylo nalezeno lepší řešení než aktuální
```

```
24              X* ⟵ X_Best ;
25        end;
26        result ⟵ X*;
27      end;
```

Algoritmus 4-12 StochasticTabuSearch

- ❖ CreatePop – viz Algoritmus 4-8, str. 28.
- ❖ $\mathrm{Sort}_{\mathrm{a}}$ - viz kapitola 1, str. 8.
- ❖ AppendList – viz kapitola 1, str. 8.
- ❖ TerminationCriterion – viz kapitola 4.1.1, str. 23.
- ❖ $\mathrm{Mutate}_{\mathrm{u}}$ – viz Algoritmus 4-10, str. 30.
- ❖ Perturbation – viz Algoritmus 4-11, str. 30.
- ❖ $\mathrm{Search}_{\mathrm{u}}$ - viz kapitola 1, str. 8.
- ❖ Length - viz kapitola 1, str. 8.
- ❖ DeleteListItem – viz kapitola 1, str. 8.

4.1.5 Stochastické simulované žíhání - Simulated Annealing

$\mathbf{X}^* \leftarrow \mathrm{StochasticSimulatedAnnealing}(\beta, E, A, B)$		
Funkce, jejímž výstupem je dosud nalezený nejlepší prvek. Generování prvku je prováděno v sousedství prvku z předchozí iterace. Horší prvek je přijat jen s jistou pravděpodobností, která je závislá na teplotě. Teplota je snižována tehdy, pokud je náhodně vygenerované číslo menší než pravděpodobnost přijetí horšího řešení. Pokud dojde ke snížení teploty je specifikovanou minimální hranici, teplota je opětovně zvýšena na počáteční specifikovanou hodnotu.		
Vstup:	β:	Nezáporná konstanta pro simulované hašení, $\beta \in (0,1)$
Vstup:	E:	Seznam obsahující délky intervalů pro jednotlivé rozhodovací proměnné – velikosti stran sousedství prvku
Vstup:	T_{Min}:	Dolní mez teploty
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	$\mathbf{X}$:	Testovaný prvek
Data:	$\mathbf{X}_{\mathrm{New}}$:	Nově vygenerovaný prvek
Data:	$F(\mathbf{X})$:	Hodnota účelové funkce prvku $\mathbf{X}$
Data:	T:	Hodnota teploty ($T \in \mathbb{R}^+$)
Data:	p:	Pravděpodobnost přijetí horšího řešení
Data:	ΔE:	Rozdíl energie - hodnot účelové funkce dvou prvků ($\Delta E \in \mathbb{R}$)
Výstup:	$\mathbf{X}^*$:	Dosud nalezený nejlepší prvek

```
1     begin
2       X ⟵ Create(A, B);  //počáteční přípustné řešení
3       X* ⟵ X;
4       T ⟵ 1;    // nastavení počáteční teploty
5       while not TerminationCriterion( ) do begin        //kritérium ukončení
6           X_New ⟵ Mutate_u(X, E);
        //mutace prvku – vygenerování nového prvku v okolí prvku X
7           X_New ⟵ Perturbation_u(X_New, A, B); //perturbace prvku
8           ΔE ⟵ F(X_New) − F(X);
9           if ΔE ≤ 0 then begin     //platí že F(X_New) ≤ F(X)
10              X ⟵ X_New;
```

```
11          if F(X) < F(X*) then X* ⟵ X ;
    (*bylo nalezeno lepší řešení než dosavadní nejlepší řešení*)
12      end
13      else begin      //platí že ΔE > 0 tj. F(X_New) > F(X)
14          p ⟵ e^-(ΔE/T); //pravděpodobnost přijetí horšího prvku
15          if Random_u( ) < p then begin
16              T ⟵ T/(1+β·T) ;   //snížení teploty pomocí vzorce simulovaného hašení
17              X ⟵ X_New ;  //přijato horší řešení
18              if T < T_Min then T ⟵ 1 ;
    //teplota klesnula pod definovanou mez, nastavení na počáteční teplotu
19          end;
20      end;
21    end;
22    result ⟵ X*;
23  end;
```

Algoritmus 4-13 StochasticSimulatedAnnealing

- Create – viz Algoritmus 4-2, str. 24.
- Perturbation – viz Algoritmus 4-11, str. 30.
- $\mathrm{Random_u}(\,)$ - funkce navrací vygenerované náhodné číslo v intervalu $[0, 1)$.

4.1.6 Stochastické lokální prohledávání - Local Search

$\mathbf{X}^* \longleftarrow \mathrm{StochasticLocalSearch}(E, A, B)$		
Funkce, jejímž výstupem je dosud nalezený nejlepší prvek. Generování prvku je prováděno v sousedství dosud nejlepšího nalezeného prvku (mutace).		
Vstup:	E:	Seznam obsahující délky intervalů pro jednotlivé rozhodovací proměnné – velikosti stran sousedství prvku
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	$\mathbf{X}$:	Nově vygenerovaný prvek pomocí mutace
Data:	$F(\mathbf{X})$:	Hodnota účelové funkce prvku $\mathbf{X}$
Data:	i:	Proměnná - čítač
Výstup:	$\mathbf{X}^*$:	Dosud nalezený nejlepší prvek

```
20  begin
21    X* ⟵ Create(A, B); //počáteční přípustné řešení
22    while not TerminationCriterion( ) do begin      //kritérium ukončení
23        X ⟵ Mutate_u(X*, E);
    (*mutace dosud nejlepšího nalezeného prvku*)
24        X ⟵ Perturbation_u(X, A, B);          //perturbace prvku
25        if F(X) < F(X*) then    //bylo nalezeno lepší řešení než dosud nejlepší nalezené řešení
26            X* ⟵ X ;
27    end;
28    result ⟵ X*;
29  end;
```

Algoritmus 4-14 StochasticLocalSearch

- ❖ Create – viz Algoritmus 4-2, str. 24.
- ❖ TerminationCriterion – viz kapitola 4.1.1, str. 23.
- ❖ Mutate_{u} – viz Algoritmus 4-10, str. 30.
- ❖ Perturbation – viz Algoritmus 4-11, str. 30.

4.1.7 Evoluční strategie - Evolution Strategy

$X^* \longleftarrow \text{ES}_{\mu+\lambda}(m, m_{\text{MP}}, q, k, A, B)$		
Algoritmus evoluční strategie - postupná vývojové strategie $(\mu + \lambda)$ - generace potomků je tvořena určitým počtem jedinců s nejlepšími funkčními hodnotami ze všech jedinců jak rodičovské populace, tak populace potomků. Generování jedince je provedeno pomocí mutace původního jedince na základě normálního rozdělení - implementováno Rechenbergovo adaptivní pravidlo jedné pětiny úspěšnosti upravující hodnoty směrodatných odchylek zvlášť pro každou dimenzi na základě vypočtené relativní četnosti úspěchů - intenzifikace vs. diverzifikace. Pokud vypočtená relativní četnost úspěchů byla shodná s definovanou konstantou, směrodatné odchylky se nemění. Přiřazení fitness jedincům je provedeno na základě setříděné populace podle hodnoty účelové funkce. V algoritmu je použita turnajová selekce.		
Vstup:	m:	Velikost populace X_{Pop} tj. délka seznamu $m = \text{Length}(X_{\text{Pop}})$
Vstup:	m_{MP}:	Počet potomků
Vstup:	q:	Počet předcházejících úspěchů, které mají být sledovány q
Vstup:	k:	Počet jedinců, mezi nimiž se uspořádá turnaj
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	n:	Počet rozhodovacích proměnných
Data:	φ:	Seznam relativních četností úspěchů o délce q
Data:	σ:	Seznam směrodatných odchylek (určuje rychlost konvergence)
Data:	X_{Pop}:	Populace rodičů (seznam), ze kterého bude proveden výběr do X_{MP}
Data:	X_{Arch}:	Populace potomků (zmutovaných rodičů) a rodičů
Data:	$\mathbf{X}$:	Vybraný jedinec, na kterém bude prováděna perturbace a mutace
Data:	$sum\varphi$:	Součet prvků v seznamu relativních četností úspěchů
Data:	i, j:	Čítače
Výstup:	X^*:	Množina nejlepších nalezených prvků

```
1    begin
2      φ ⟵ ( );
3      σ ⟵ ( );
4      n ⟵ min{Length(A),Length(B)};     //získání počtu rozhodovacích prom.
5      for j ⟵ 0 to n − 1 do
       (*vytvoření počátečního seznamu směrodatných odchylek = 1/3 velikosti X̃_j*)
6          σ ⟵ AddListItem(σ, |(B[j] − A[j])/3|);
7      X_Pop ⟵ CreatePop(m, A, B);       //vytvoření počáteční populace
8      while not TerminationCriterion( ) do begin       //kritérium ukončení
9          X_Arch ⟵ ( );
10         for i ⟵ 0 to m_MP − 1 do   begin
11             X ⟵ Mutate_n(X_Pop[i mod m], σ);
       (*mutace jedince na základě normálního rozdělení*)
12             X ⟵ Perturbation(X, A, B);
       (*pokud by nový zmutovaný jedinec ležel mimo prohledávaný prostor*)
```

```
13          if Length(φ) ≥ q then
    (*pokud seznam četností úspěchů dosáhnul specifikované délky, bude umazán prvek na první pozici
    v seznamu relativních četností úspěchů* )
14              φ ← DeleteListItem(φ, 0);
15          if CF_F(X)(X, X_Pop[i mod m]) < 0 then
    //jedinec X je lepší, než jedinec z X_Pop na pozici [i mod m]
16              φ ← AddListItem(φ, 1);
    (*potomek je lepší než rodič tzn., bude přidána 1 na konec seznamu četností úspěchů*)
17          else φ ← AddListItem(φ, 0);
18          sumφ ← 0;
19          for i ← 0 to Length(φ) − 1 do
20              sumφ ← sumφ + φ[i];
21          if (sumφ/Length(φ)) < 0.2 then
22              for j ← 0 to n − 1 do
23                  σ[j] ← σ[j] ∗ 0.82
24              else  if (sumφ/Length(φ)) > 0.2 then
25                      for j ← 0 to n − 1 do
26                          σ[j] ← σ[j] ∗ 1.22;
27          X_Arch ← AddListItem(X_Arch, X);  //spojení populace rodičů a potomků
28      end;
29      X_Pop ← AppendList(X_Pop, X_Arch);
30      AssignFitnessRank(X_Pop);
    (*přiřazení fitness na základě setříděné populace podle hodnoty účelové funkce (na základě
    komparační funkce CF_F(X)) – minimalizace*)
31      X_Pop ← TournamentSelect_r(X_Pop, m, k);
    (*turnajová selekce*)
32    end;
33    result ← ExtractOptimalSet(X_Pop);
34  end;
```

Algoritmus 4-15 $\mathbf{ES_{\mu+\lambda}}$

- min – viz kapitola 4.1.1, str. 24.
- AddListItem – viz kapitola 1, str. 8.
- CreatePop – viz Algoritmus 4-8, str. 28.
- TerminationCriterion – viz kapitola 4.1.1, str. 23.
- Perturbation – viz Algoritmus 4-11, str. 30.
- Length - viz kapitola 1, str. 8.
- DeleteListItem - viz kapitola 1, str. 8.
- AppendList - viz kapitola 1, str. 8.
- AssignFitnessRank – viz Algoritmus 3-3, str. 12.
- $\text{TournamentSelect}_r$ – viz Algoritmus 3-11, str. 19.
- ExtractOptimalSet- viz Algoritmus 4-6, str. 26.

$\mathbf{X}_{\text{Mut}} \leftarrow \text{Mutate}_{\text{n}}(\mathbf{X}, \sigma)$	
Funkce, jejímž výstupem je zmutovaný prvek, který vznikl transformací (v kontextu evolučních algoritmů mutací) vstupního prvku $\mathbf{X}$. Transformace jednotlivých složek vstupního prvku se provádí na základě vygenerování náhodné čísla generovaného podle normálního rozdělení se směrodatnou odchylkou definovanou pro každou jednotlivou rozhodovací proměnnou (geometricky pro každou osu).	
Vstup:	$\mathbf{X}$: Původní prvek, jehož souřadnice budou transformovány – seznam, jehož prvky budou transformovány
Vstup:	σ: Seznam obsahující směrodatné odchylky pro jednotlivé rozhodovací proměnné

Data:	j:	Proměnná - čítač
Výstup:	$\mathbf{X}_{\mathrm{Mut}}$:	Prvek - seznam, jehož složky jsou zmutované - transformované

```
1   begin
2     Randomize;
3     X_Mut ⟵ ( );
4     for j ⟵ 0 to (Length(X) − 1) do
5         X_Mut ⟵ AddListItem(X_Mut, Random_n(X[j], σ[j]));
      (*přidání hodnoty souřadnice do vektoru X_Mut pomocí funkce pro generování náhodného čísla
      podle normálního rozdělení*)
6     result ⟵ X_Mut;
7   end;
```

Algoritmus 4-16 $\mathbf{Mutate}_n$- transformace složek prvku na základě normálního rozdělení

4.1.8 Diferenciální evoluce - Differential Evolution

$X^* \leftarrow \mathrm{DE}(m, CR, \omega_{\min}, A, B)$		
Algoritmus diferenciální evoluce – selekce je provedena mezi rodičem a jeho potomkem, který vznikl křížením rodiče s nově vytvořeným jedincem vzniklým mutací jedinců pomocí metody BEST. Lepší jedinec, z těchto dvou porovnávaných jedinců, je následně zařazen do populace, která kompletně nahradí populaci rodičů. V algoritmu je uplatněno adaptivní pravidlo podle Ali a Törna, která upravuje hodnotu mutačního parametru ω. Dále je definována pravděpodobnost CR určená pro křížení, tzn., že potomek zdědí gen zmutovaného jedince, nebo zdědí gen po rodiči.		
Vstup:	m:	Velikost populace rodičů X_{Pop} tj. délka seznamu $m = \mathrm{Length}(X_{\mathrm{Pop}})$, doporučená hodnota $m = 10 \cdot n$
Vstup:	CR:	Pravděpodobnost nahrazení souřadnic rozhodovacích proměnných, $CR \in [0,1]$
Vstup:	$\omega_{\min}$:	Konstanta adaptivního pravidla, $\omega_{\min} > 0$, doporučená hodnota $\omega_{\min} \geq 0.45$
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných
Vstup:	B:	Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	m_{MP}:	Počet jedinců, kteří budou umístěni do výsledného seznamu určeného k páření
Data:	ω:	Parametr adaptivního pravidla, $\omega \in [\omega_{\min}, 1)$
Data:	n:	Počet rozhodovacích proměnných
Data:	X_{Pop}:	Populace rodičů (seznam)
Data:	X_{Arch}:	Populace sloužící jako proměnná
Data:	X_{MP}:	Populace potomků
Data:	$\mathbf{X}$:	Vybraný jedinec
Data:	$\mathbf{X}_{\mathrm{New}}$:	Zmutované jedinec pro křížení
Data:	b:	Logická proměnná typu boolean - informace, zda byla alespoň jedna souřadnice rozhodovací proměnné potomka přepsána souřadnicí rozhodovací proměnné nově vygenerovaným jedincem pro křížení
Data:	r:	Náhodně zvolený index rozhodovací proměnné potomka, který bude přepsán souřadnicí rozhodovací proměnné nově vygenerovaným jedincem pro křížení
Data:	$F_{\max}$:	Největší hodnota účelové funkce jedince, ze všech jedinců v populaci
Data:	$F_{\min}$:	Nejmenší hodnota účelové funkce jedince, ze všech jedinců v populaci
Data:	i, j:	Čítače
Výstup:	X^*:	Množina nejlepších nalezených prvků

```
1   begin
```

```
2     m_MP ← m;
    (*velikost populace rodičů je stejná jako velikost populace potomků*)
3     ω ← ω_min ;  //počáteční nastavení parametru adaptivního pravidla
4     n ← min{Length(A),Length(B)};//získání počtu rozhodovacích prom.
5     X_Pop ← CreatePop(m, A, B);      //vytvoření počáteční populace
6     while not TerminationCriterion( ) do begin      //kritérium ukončení
7        for i ← 0 to m_MP − 1 do  begin
8           X ← X_Pop[i]; //vybraný jedinec z populace
9           X_New ← DEBest(X_Pop, X, n, ω, A, B) ;
    (*jedinec vzniklý mutací 4 jedinců pro následné křížení s vybraným jedincem X *)
10             b ← False;
11          for j ← 0 to n − 1 do
12             if (Random_u( ) ≤ CR) then begin
13                X[j] ← X_New[j];     //vznik nového potomka křížením
14                b ← True;
15             end;
16          if (b = False) then begin
    (*během křížení nedošlo k náhradě alespoň jedné ze souřadnic rozhodovací proměnné*)
17             r ← ⌊Random_u(0,Length(n))⌋;
18             X[r] ← X_New[r];
19          end;
20          if (CF_F(X)(X, X_Pop[i]) < 0) then
    (*selekce lepšího jedince – potomek vs. rodič*)
21             X_MP ← AddListItem(X_MP, X);
22          X_MP ← Sort_a(X_MP, CF_F(X));
23          F_min ← F(X_MP[0]);
24          F_max ← F(X_MP[Length(X_MP) − 1]);
25          if (|F_max/F_min| < 1) then  //adaptivní pravidlo
26             ω ← max{ω_min, 1 − |F_max/F_min|}
27          else ω ← max{ω_min, 1 − |F_min/F_max|};
28       end;
29       X_Pop ← X_MP;
30    end;
31    result ← ExtractOptimalSet(X_Pop);
32 end;
```

Algoritmus 4-17 DE

$\mathbf{X}_{\mathrm{New}} \longleftarrow \mathrm{DEBest}(X_{\mathrm{Pop}}, \mathbf{X}, n, \omega, A, B)$		
Funkce, která generuje nového jedince na základě náhodně čtyř vybraných jedinců, které jsou různí od nejlepšího jedince v populaci a vybraného jedince z populace $\mathbf{X}$ – postup označovaný jako BEST. Jedinec vznikl mutací.		
Vstup:	X_{Pop}:	Populace rodičů (seznam)
Vstup:	$\mathbf{X}$:	Vybraný jedinec z populace rodičů
Vstup:	n:	Rozměr prohledávaného prostoru
Data:	ω:	Parametr adaptivního pravidla, $\omega \in [\omega_{\min}, 1)$
Vstup:	A:	Seznam dolních mezí prohledávaných intervalů rozhodovacích proměnných

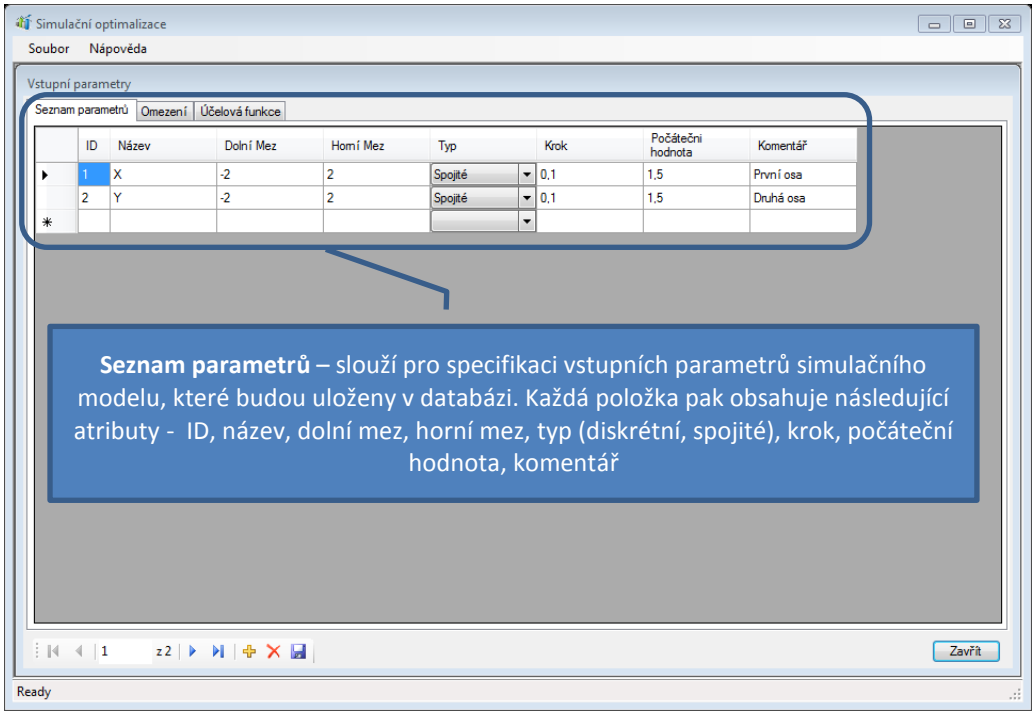
Vstup:	B: Seznam horních mezí prohledávaných intervalů rozhodovacích proměnných
Data:	X_{Arch}: Populace sloužící jako proměnná
Data:	$\mathbf{X}_{\mathrm{Best}}$: Nejlepší jedinec v populaci
Data:	$\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$: První, druhý, třetí, čtvrtý vybraný jedinec v populaci k vygenerování nového prvku
Data:	j: Proměnné – čítače
Výstup:	$\mathbf{X}_{\mathrm{New}}$: Nově vygenerovaný prvek

```
1   begin
2     X_Arch ⟵ X_Pop ;
3     X_Arch ⟵ Sort_a(X_Arch, CF_F(X));
    (*vzestupné třídění populace na základě účelové funkce – minimalizace účelové funkce*)
4     X_Best ⟵ X_Arch[0];   //nejlepší jedinec v populaci
5     X_Arch ⟵ DeleteListItem(X_Arch, 0);
    (*odstranění nejlepšího jedince z populace*)
6     if Search_u(X, X_Arch) ≥ 0 then //vybraný jedinec X nebyl již odstraněn
7       X_Arch ⟵ DeleteListItem(X_Arch, Search_u(X, X_Arch));
8     X_Arch ⟵ RandomSelect_w(X_Arch, 4);
    (*výběr čtyř navzájem odlišných prvků různých od X_Best a X*)
9     X_0 ⟵ X_Arch[0];
10    X_1 ⟵ X_Arch[1];
11    X_2 ⟵ X_Arch[2];
12    X_3 ⟵ X_Arch[3];
13    for j ⟵ 0 to n − 1 do
14      X_New[j] ⟵ X_Best[j] + ω ∗ (X_0[j] + X_1[j] − X_2[j] − X_3[j]);    //mutace
15    X_New ⟵ Perturbation(X_New, A, B);   //perturbace – zrcadlení prvku
16    Result ⟵ X_New ;
17  end;
```

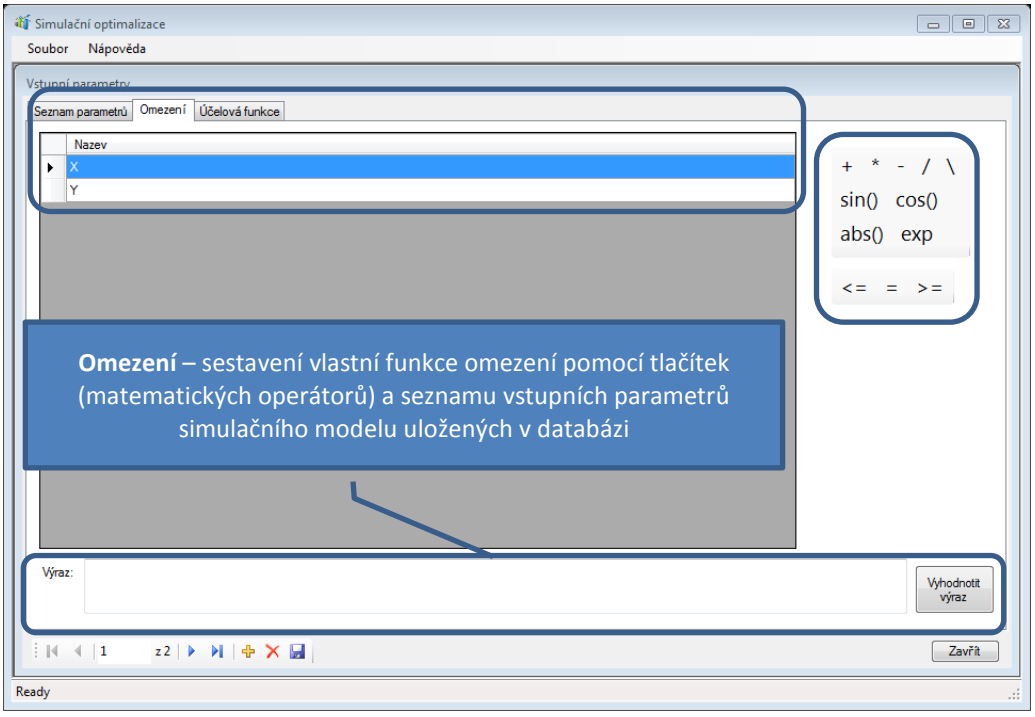
Algoritmus 4-18 DEBest

4.2 Popis prostředí softwarové aplikace simulační optimalizace

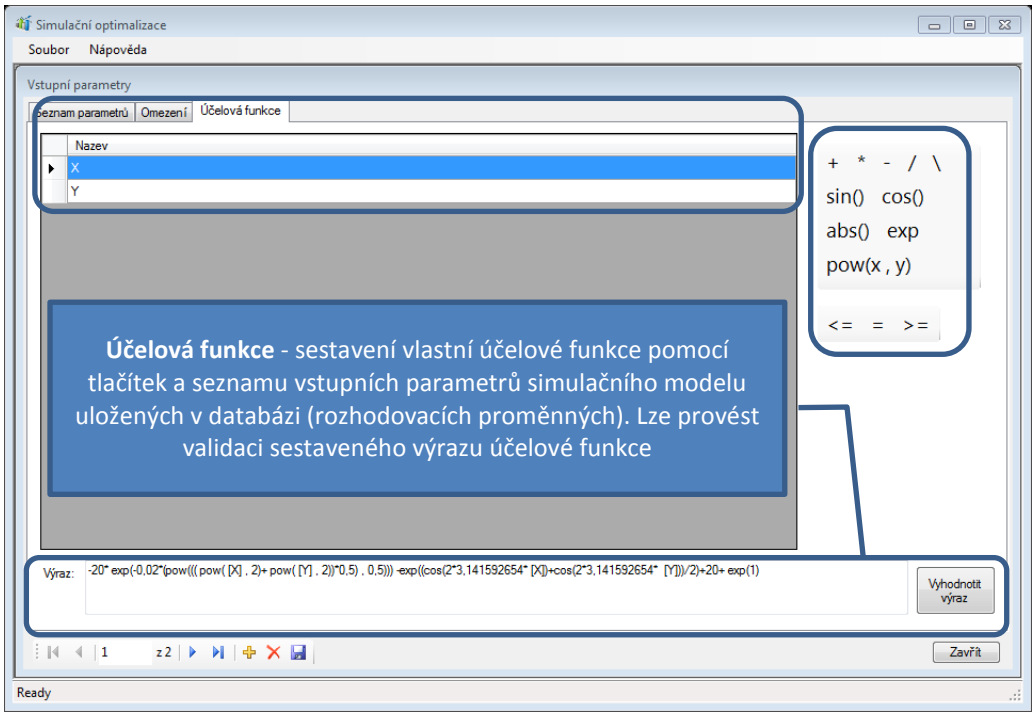
4.2.1 Vstupní parametry



Obr. 4-1 Snímek prostředí pro specifikaci rozhodovacích proměnných

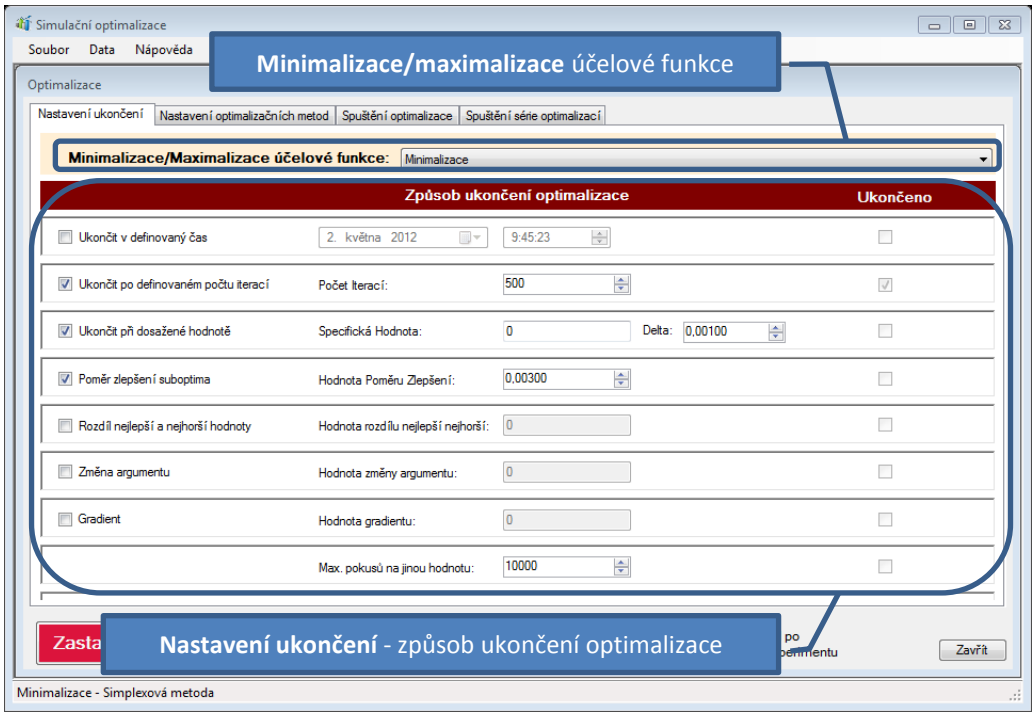


Obr. 4-2 Snímek prostředí pro specifikaci omezení

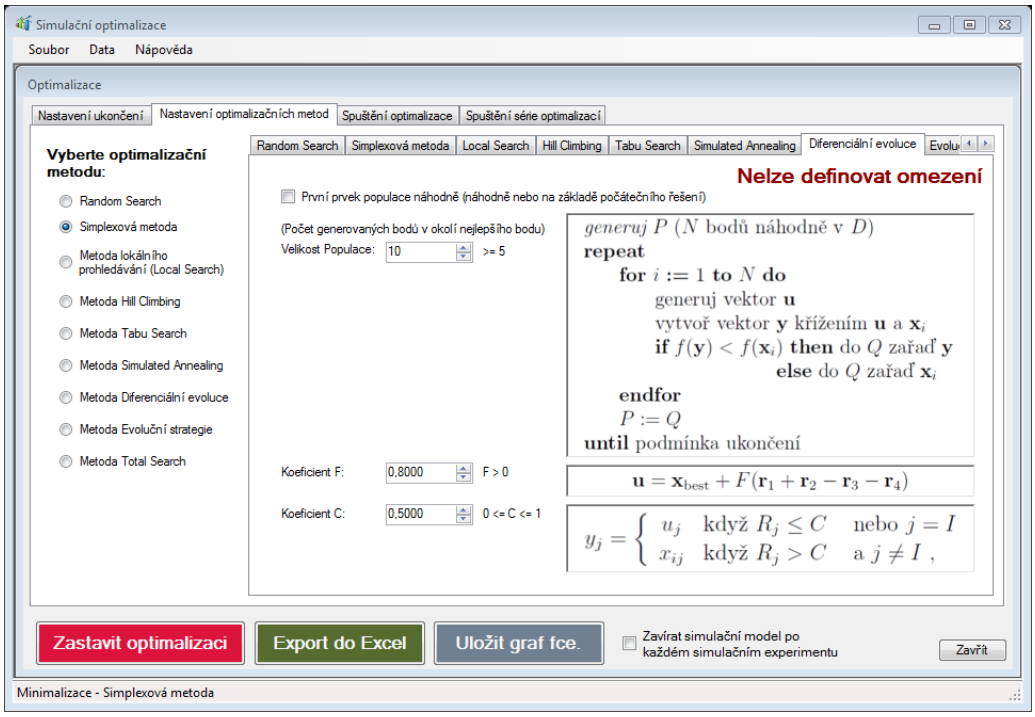


Obr. 4-3 Snímek prostředí pro specifikaci účelové funkce

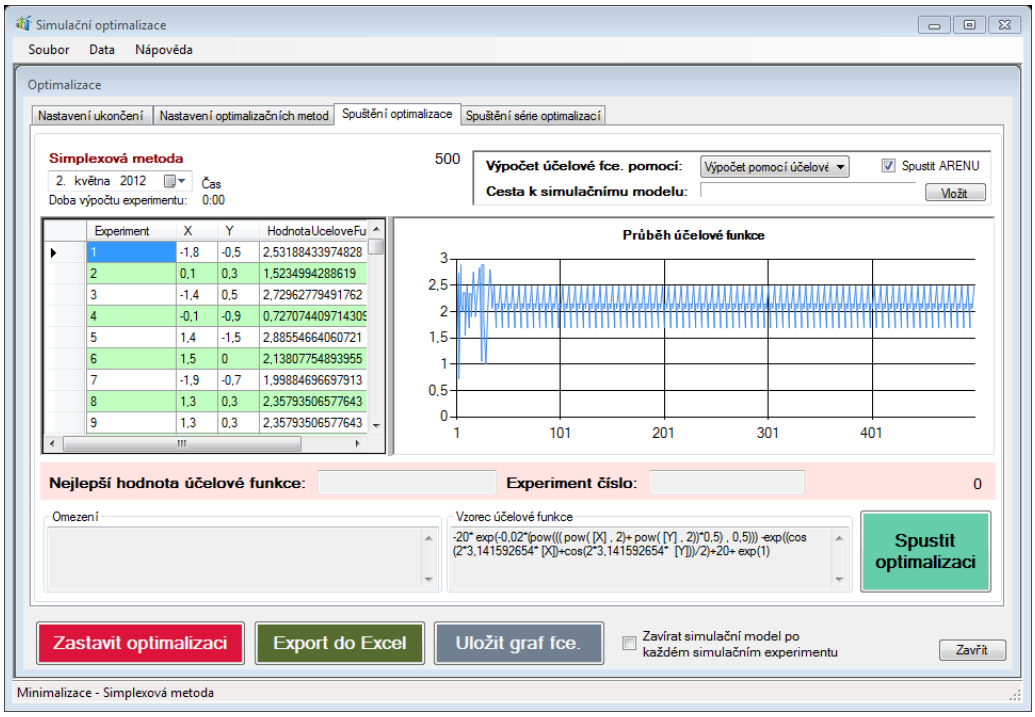
4.2.2 Nastavení optimalizace



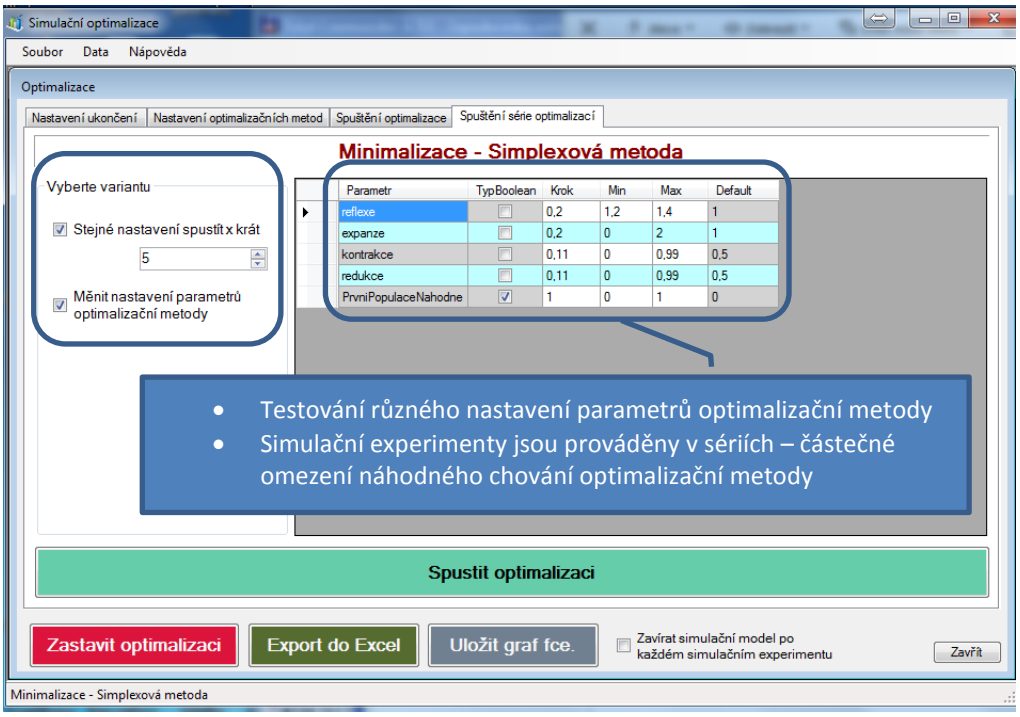
Obr. 4-4 Snímek prostředí pro specifikaci minimalizace/maximalizace účelové funkce a nastavení ukončení



Obr. 4-5 Snímek prostředí pro výběr optimalizačního algoritmu a nastavení jeho parametrů

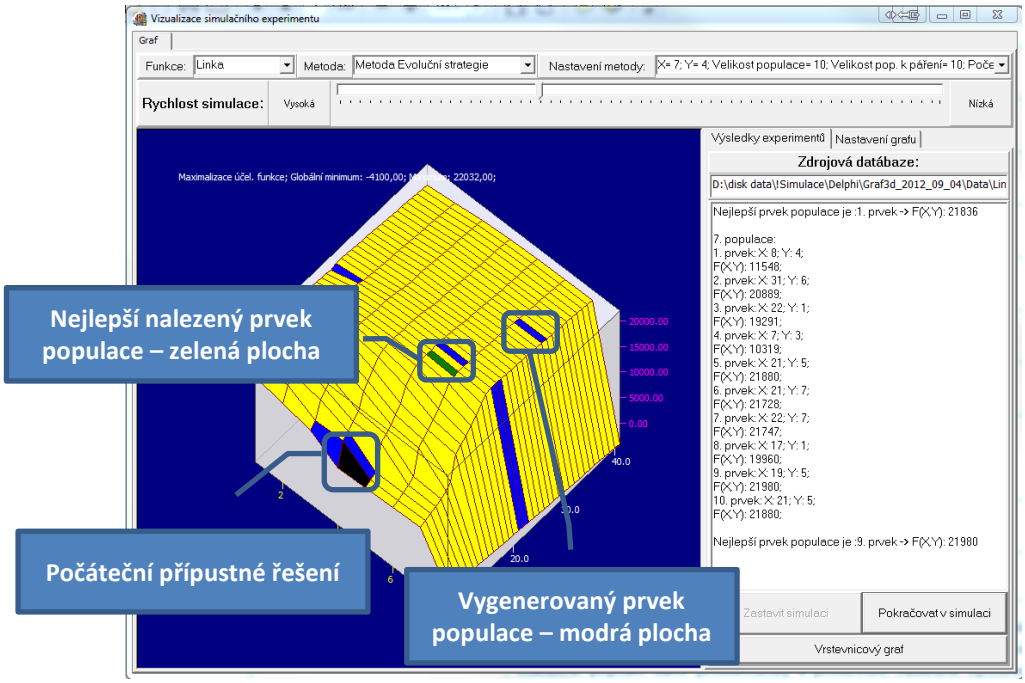


Obr. 4-6 Snímek prostředí záložky „Spuštění optimalizace“

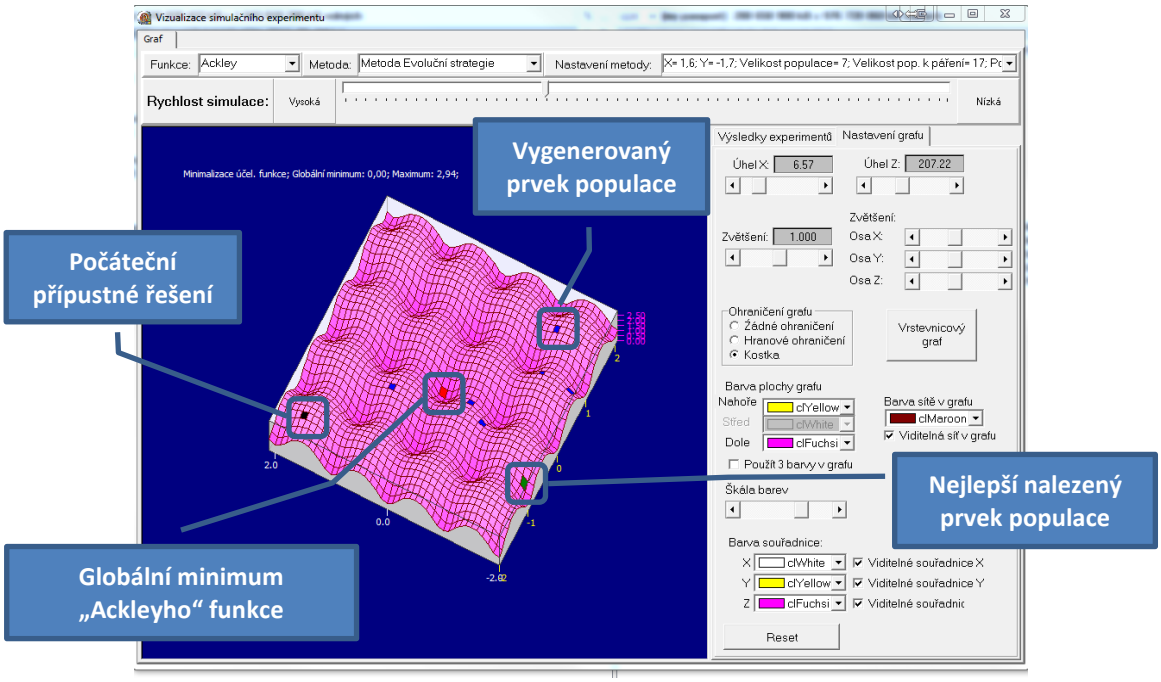


Obr. 4-7 Snímek záložky „Spuštění série optimalizací“

4.3 Softwarová aplikace Vizualizace



Obr. 4-8 Snímek softwarové aplikace Vizualizace optimalizačního experimentu – optimalizační algoritmus Evoluční strategie – simulační model „VyrobniLinka“



Obr. 4-9 Snímek softwarové aplikace Vizualizace optimalizačního experimentu – nastavení grafu účelové funkce – optimalizační algoritmus Evoluční strategie – simulační model „Ackley“

5 Simulační modely

5.1 Simulační model „De Jong“

5.1.1 Popis simulačního modelu

Simulační model charakterizuje De Jongovu funkci. Tato konvexní funkce představuje *n*-rozměrnou parabolu. Funkce je snadnou úlohou pro optimalizační metody. Jedná se o minimalizaci účelové funkce. Globální minimum De Jongovi funkce v dvourozměrném prohledávaném prostoru je $F(\breve{\mathbf{X}}) = F(0.0{,}0.0) = 0, \breve{\mathbf{X}} \in \tilde{X}$. Toto jediné globální minimum je také zároveň i globálním optimem v prostoru všech řešení.

5.1.2 Rozhodovací proměnné – vstupní parametry simulačního modelu

Jako rozhodovací proměnné jsou nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	První osa prohledávaného prostoru
	X_2	Druhá osa prohledávaného prostoru

Tabulka 5-1 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Poznámka: Indexy rozhodovacích proměnných začínají jedničkou z důvodu vykreslení první rozhodovací proměnné v trojrozměrném grafu účelové funkce jako první osy prohledávaného prostoru. V případě seznamů by byl první index rozhodovací proměnné roven nule a poslední index $n - 1$.

Prohledávaný prostor:

Prohled. prostor	Počet prvků v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	1681	X_1	-2	2	0.1	2
		X_2	-2	2	0.1	2

Tabulka 5-2 Specifikace prohledávaného prostoru

5.1.3 Účelová funkce

Účelová funkce je vyjádřena analyticky:

Účelová funkce	Popis	Minimalizace účelové funkce
$F(\mathbf{X})$	De Jongova funkce	Ano

Tabulka 5-3 Specifikace účelové funkce

Předpis účelové funkce:

$$F(\mathbf{X}) = \sum_{j=1}^{n} {x_j}^2 \tag{5.1}$$

kde:

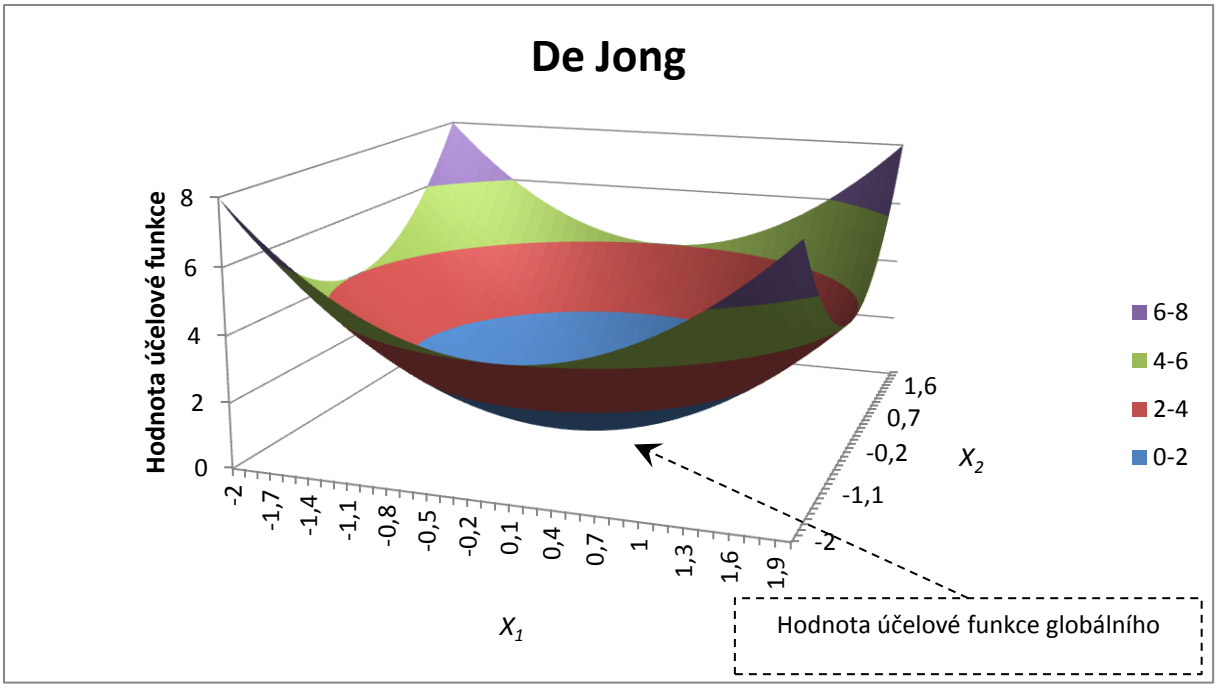
- $F(\mathbf{X})$ … Účelová funkce.
- n … Počet dimenzí prostoru.
- j … Index rozhodovací proměnné v prohledávaném prostoru.
- x_j … Hodnota souřadnice bodu – prvku rozhodovací proměnné.

Globální maxima účelové funkce v prohledávaném prostoru jsou čtyři – v každém rohu prohledávaného prostoru.

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	0	0	2	8
		X_2	0		2	
		X_1	0	0	-2	8
		X_2	0		-2	
		X_1	0	0	-2	8
		X_2	0		2	
		X_1	0	0	2	8
		X_2	0		-2	

Tabulka 5-4 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-1) je zobrazen průběh De Jongovi funkce ve formě dvourozměrného paraboloidu. Globální minimum je uprostřed ve středu údolí prohledávaného prostoru.



Obr. 5-1 Průběh účelové funkce simulačního modelu – De Jongovi funkce

5.1.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	1681	0	0.001	
Downhill Simplex, Diferenciální evoluce	500	0	0.001	0.003

Tabulka 5-5 Specifikovaná kritéria ukončení

Hodnota kritéria ukončení „Ukončit po definovaném počtu iterací“ bylo stanovena tak, aby optimalizační metoda měla možnost otestovat všechny prvky vzhledem k poměrně malému prohledávanému prostoru. U metod Downhill Simplex a Diferenciální evoluce byl v kritériu ukončení po definovaném počtu iterací zvolen záměrně menší počet provedení simulačních experimentů z důvodu rychlé konvergence.

5.1.5 Čas simulačního experimentu

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu [sec]
De Jong	0.2

Tabulka 5-6 Čas potřebný k proběhnutí simulačního experimentu

Do času je započteno spočtení hodnoty účelové funkce prvku a uložení výsledku do databáze. Simulační experimenty byly prováděny na počítači o parametrech:

- Procesor (CPU): Intel(R) Core(TM)2 Duo CPU, E6750 @2.66GHz
- Paměť (RAM): 2.67 GHz, 1.98 GB RAM
- Operační systém: Microsoft Windows XP

5.2 Simulační model „Rosenbrock“

5.2.1 Popis simulačního modelu

Simulační model charakterizuje Rosenbrockovu (též označovanou „banánovou“, „sedlovou“ nebo druhou De Jongovu) funkci. Tato nekonvexní funkce představuje dlouhé, úzké, parabolické ploché údolí. Funkce je středně obtížnou úlohou pro optimalizační metody. Jedná se o minimalizaci účelové funkce. Globální minimum Rosenbrockovi funkce v dvourozměrném definovaném prohledávaném prostoru je $F(\check{\mathbf{X}}) = F(1.0,1.0) = 0, \check{\mathbf{X}} \in \tilde{X}$. Toto jediné globální minimum je také zároveň i globálním optimem v prostoru všech řešení.

5.2.2 Rozhodovací proměnné

Jako rozhodovací proměnné nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	První osa prohledávaného prostoru
	X_2	Druhá osa prohledávaného prostoru

Tabulka 5-7 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	1681	X_1	-2	2	0.1	-2
		X_2	-2	2	0.1	-2

Tabulka 5-8 Specifikace prohledávaného prostoru

5.2.3 Účelová funkce

Účelová funkce je vyjádřena analyticky:

Účelová funkce	Popis	Minimalizace účelové funkce
$F(\mathbf{X})$	Rosenbrockova funkce	Ano

Tabulka 5-9 Specifikace účelové funkce

Předpis účelové funkce: [14]

$$F(\mathbf{X}) = \sum_{j=1}^{n-1} 100 \cdot \left(x_j{}^2 - x_{j+1}\right)^2 + \left(1 - x_j\right)^2 \tag{5.2}$$

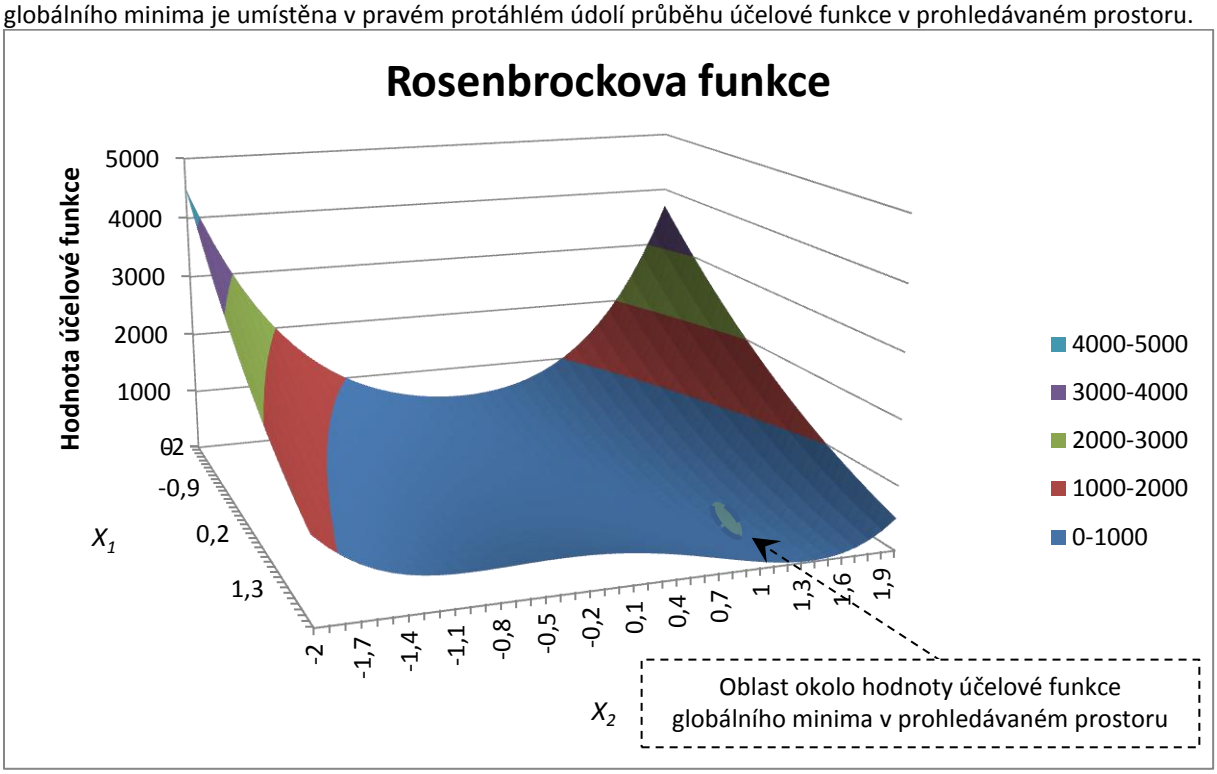
kde:

- $F(\mathbf{X})$ … Účelová funkce.
- n … Počet dimenzí prostoru.
- j … Index rozhodovací proměnné v prohledávaném prostoru.
- x_j … Hodnota souřadnice bodu – prvku rozhodovací proměnné.

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	1	0	-2	4500
		X_2	1		-2	

Tabulka 5-10 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-2) je zobrazen průběh Rosenbrockovi funkce. Hodnota účelové funkce globálního minima je umístěna v pravém protáhlém údolí průběhu účelové funkce v prohledávaném prostoru.



Obr. 5-2 Průběh účelové funkce simulačního modelu – Rosenbrockovi funkce

5.2.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	1681	0	0.001	
Downhill Simplex, Diferenciální evoluce	500	0	0.001	0.003

Tabulka 5-11 Specifikovaná kritéria ukončení

5.2.5 Simulační experimenty

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu (t[sec])
Rosenbrock	0.2

Tabulka 5-12 Čas potřebný k proběhnutí simulačního experimentu

5.3 Simulační model „Michalewicz“

5.3.1 Popis simulačního modelu

Simulační model charakterizuje Michalewiczovu funkci. Tato funkce představuje n-rozměrnou multimodální účelovou funkci (obsahující $n!$ lokálních optim). Funkce je těžkou úlohou pro optimalizační metody. Jedná se o minimalizaci účelové funkce. Globální minimum Michalewiczovi funkce v dvourozměrném definovaném prohledávaném prostoru je $F(\check{\mathbf{X}}) = F(2.2,1.6) = -1.78716856945323, \check{\mathbf{X}} \in \tilde{X}$. Hodnota globálního minima závisí na konkrétní hodnotě parametrů funkce.

5.3.2 Rozhodovací proměnné

Jako rozhodovací proměnné jsou nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	První osa prohledávaného prostoru
	X_2	Druhá osa prohledávaného prostoru

Tabulka 5-13 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Prohledávaný prostor je obvykle omezen na hyperkostku o rozměru $0 \leq x_j \leq \pi$. Pokud prostor obsahuje pět dimenzí ($n = 5$), lze hodnotu globálního minima aproximovat hodnotou $F(\check{\mathbf{X}}) = -4.687$. Při 10 dimenzích prostoru je hodnota účelová funkce aproximována hodnotou $F(\check{\mathbf{X}}) = -9.66$. [43]

Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	961	X_1	0	3	0.1	0
		X_2	0	3	0.1	0

Tabulka 5-14 Specifikace prohledávaného prostoru

5.3.3 Účelová funkce

Účelová funkce je vyjádřena analyticky:

Účelová funkce	Popis	Minimalizace účelové funkce
$F(\mathbf{X})$	Michalewiczova funkce	Ano

Tabulka 5-15 Specifikace účelové funkce

Předpis účelové funkce: [32]

$$F(\mathbf{X}) = -\sum_{j=1}^{n} \sin(x_j) \cdot \left(sin\left(\frac{j \cdot {x_j}^2}{\pi} \right) \right)^{2 \cdot m} \tag{5.3}$$

kde:

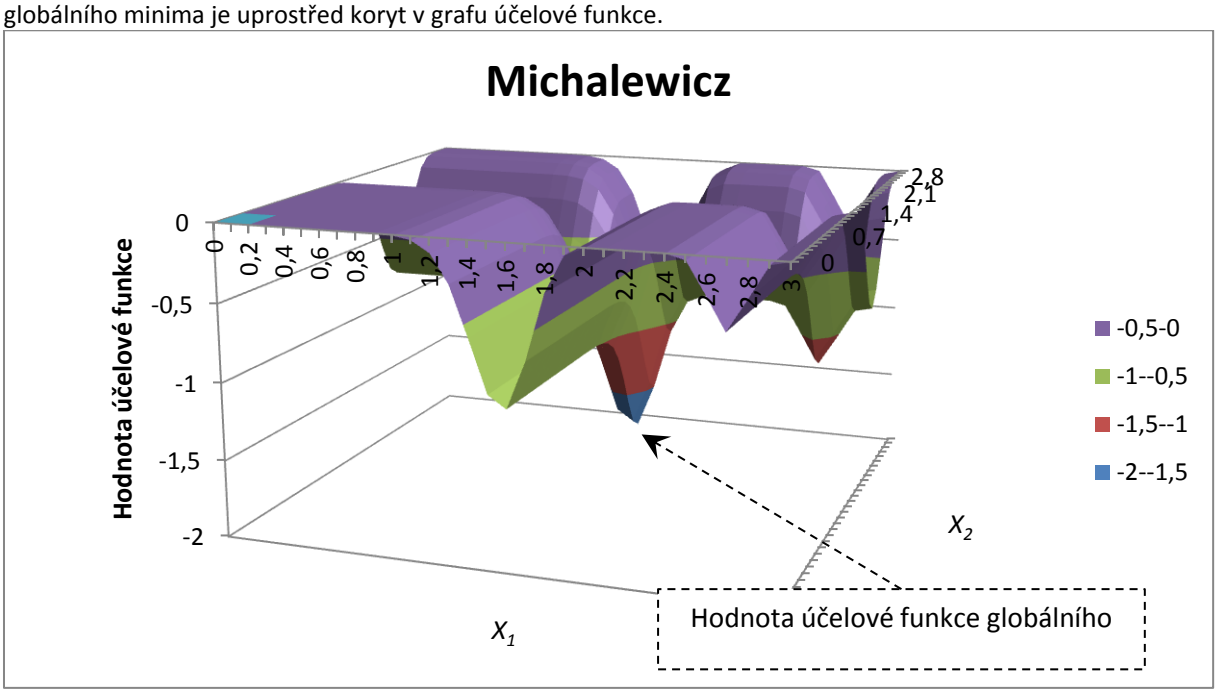
- $F(\mathbf{X})$ … Účelová funkce.
- n … Počet dimenzí prostoru.
- j … Index rozhodovací proměnné v prohledávaném prostoru.
- x_j … Hodnota souřadnice bodu – prvku rozhodovací proměnné.
- m … Parametr ovlivňující příkrost údolí a hran. Pro naši účelovou funkci je stanoveno $m = 5$.

Pro účelovou funkci bylo stanoveno $m = 5$. Parametr m stanovuje příkrost údolí a hran. Větší m vede k těžšímu prohledávání účelové funkce. Pro velké m funkce vypadá jako jehla v kupce sena (hodnoty účelové funkce bodů mimo úzké vrcholky poskytují velmi málo informací o pozici globálního optima).

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum ($\check{\mathbf{X}}$) účelové funkce v prohledávaném prostoru		Globální maximum ($\hat{\mathbf{X}}$) účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	2.2	-1,787168569453	0	0
		X_2	1.6		0	

Tabulka 5-16 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-3) je zobrazen průběh Michalewiczovi funkce. Hodnota účelové funkce globálního minima je uprostřed koryt v grafu účelové funkce.



Obr. 5-3 Průběh účelové funkce simulačního modelu – Michalewiczovi funkce

5.3.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	961	-1,78716856945323	0.001	
Downhill Simplex, Diferenciální evoluce	500	-1,78716856945323	0.001	0.003

Tabulka 5-17 Specifikovaná kritéria ukončení

5.3.5 Simulační experimenty

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu (t[sec])
Michalewicz	0.2

Tabulka 5-18 Čas potřebný k proběhnutí simulačního experimentu

5.4 Simulační model „Ackley“

5.4.1 Popis simulačního modelu

Simulační model charakterizuje Ackleyho funkci. Tato funkce je multimodální – obsahuje mnoho vrcholů. Funkce je dobrým testem předčasné konvergence („zaseknutí“ v lokálním optimu) pro optimalizační metodu. Jedná se o minimalizaci účelové funkce. Globální minimum Ackleyho funkce v dvourozměrném prohledávaném prostoru je $F(\breve{\mathbf{X}}) = F(0.0,0.0) = 0, \breve{\mathbf{X}} \in \tilde{X}$. Toto globální minimum je také zároveň i globálním optimem v prostoru všech řešení.

5.4.2 Rozhodovací proměnné – vstupní parametry simulačního modelu

Jako rozhodovací proměnné jsou nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	První osa prohledávaného prostoru
	X_2	Druhá osa prohledávaného prostoru

Tabulka 5-19 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Poznámka: Indexy rozhodovacích proměnných začínají jedničkou z důvodu vykreslení první rozhodovací proměnné v trojrozměrném grafu účelové funkce jako první osy prohledávaného prostoru. V případě seznamů by byl první index rozhodovací proměnné roven nule a poslední index $n - 1$.

Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	1681	X_1	-2	2	0.1	1.5
		X_2	-2	2	0.1	1.5

Tabulka 5-20 Specifikace prohledávaného prostoru

5.4.3 Účelová funkce

Účelová funkce je vyjádřena analyticky:

Účelová funkce	Popis	Minimalizace účelové funkce
$F(\mathbf{X})$	Ackleyho funkce	Ano

Tabulka 5-21 Specifikace účelové funkce

Předpis účelové funkce: [14]

$$F(\mathbf{X}) = -20 \cdot \exp\left(-0.02 \cdot \sqrt{\frac{1}{n} \cdot \sum_{j=1}^{n} {x_j}^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{j=1}^{n} \cos 2\pi x_j\right) + 20 + \exp(1) \tag{5.4}$$

kde:

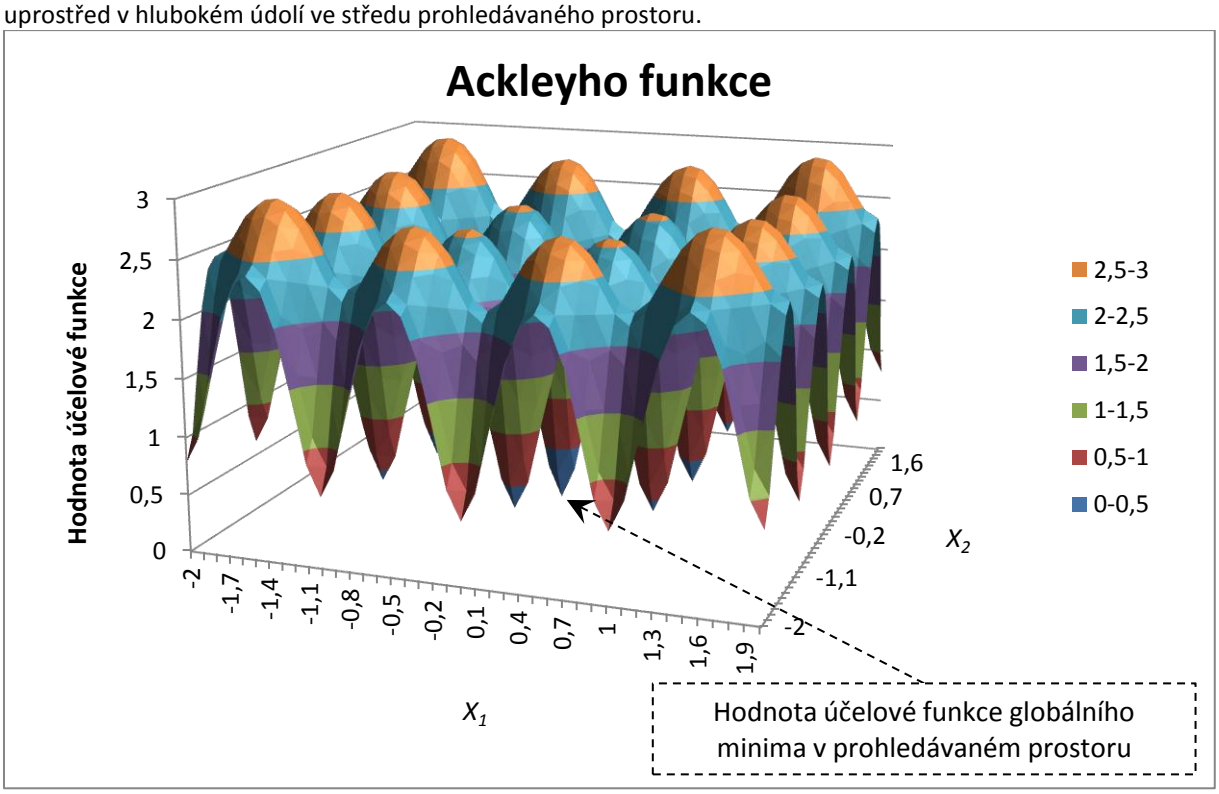
- $F(\mathbf{X})$ … Účelová funkce.
- n … Počet dimenzí prostoru.
- j … Index rozhodovací proměnné v prohledávaném prostoru.
- x_j … Hodnota souřadnice bodu – prvku rozhodovací proměnné.
- $\exp(1) = e^1$ … Vrátí *e* umocněné na hodnotu argumentu – číslo v závorce.

Globální maxima účelové funkce v prohledávaném prostoru jsou čtyři ve formě čtyř vrcholků v každém rohu prohledávaného prostoru.

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	0	0	1.5	2.941491716
		X_2	0		1.5	
		X_1	0	0	-1.5	2.941491716
		X_2	0		-1.5	
		X_1	0	0	-1.5	2.941491716
		X_2	0		1.5	
		X_1	0	0	1.5	2.941491716
		X_2	0		-1.5	

Tabulka 5-22 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-4) je zobrazen průběh Ackleyho multimodální funkce. Globální minimum je uprostřed v hlubokém údolí ve středu prohledávaného prostoru.



Obr. 5-4 Průběh účelové funkce simulačního modelu – Ackleyho funkce

5.4.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	1681	0	0.001	
Downhill Simplex, Diferenciální evoluce	500	0	0.001	0.003

Tabulka 5-23 Specifikovaná kritéria ukončení

U metod Downhill Simplex a Diferenciální evoluce byl v kritériu ukončení po definovaném počtu iterací zvolen záměrně menší počet provedení simulačních experimentů z důvodu rychlé konvergence.

5.4.5 Simulační experiment

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu [sec]
Ackley	0.2

Tabulka 5-24 Čas potřebný k proběhnutí simulačního experimentu

Do času je započteno spočtení hodnoty účelové funkce prvku a uložení výsledku do databáze.

5.5 Simulační model „Doprava“

5.5.1 Popis simulačního modelu

Simulační model zachycuje celkovou interní logistiku ve výrobní hale a skladu. Doprava je zajišťována pomocí vláčků, ke kterým se zapojují vagóny. V modelu jsou definovány jednotlivé dopravní cesty, které jsou zachyceny v následujících obrázcích. Při navážení materiálů ze skladu vláček dle potřeby zaváží předmontážní pracoviště a všechny montážní linky. Trasa, kterou vláček projede, je odvozena od cílových míst přepravovaného materiálu. Některé dílce jsou přepravovány z předvýroby do skladu.

Existují následující skupiny vláčků, které zajišťují pouze dopravu:

1. Doprava malých dílců pro montážní linky a pro předmontáže v předvýrobě.
2. Doprava velkých dílců pro montážní linky.
3. Odvoz hotových výrobků od montážních linek a navážení obalových materiálů.

Hodnota účelové funkce globálního maxima účelové funkce v třírozměrném prohledávaném prostoru je $F(\mathbf{X}) = F(11.0; 24.0; 11.0) \approx 22.65, \mathbf{X} \in \tilde{X}$.

5.5.1.1 Doprava malých dílců pro montážní linky a pro předmontáže v předvýrobě

Na následujícím obrázku (Obr. 5-5 Sklad – doprava malých dílců) jsou znázorněny dopravní cesty nutné pro dopravu malých dílců. Modře jsou znázorněny cesty, kdy na vagónu jsou naložené bedničky s malými dílci. Zeleně jsou znázorněny cesty, kdy se vrací vagóny s prázdnými bedničkami z výroby (montážní linky, předmontáže), nebo se vrací tzv. vrácenky od montážních linek.

Popis chování dopravy ze skladu pro malé dílce k montážní lince:

- Vyskladnění z regálového skladu se modeluje pomocí kapacity zakladače a doby vyskladnění jedné bedničky
- Bedničky se pokládají na podvozek vagónu (pouze bedničky určené na stejnou montážní linku, nebo na stejnou předmontáž).
- Vyskladňovat lze pouze i jednu bedničku. Tzn., že se nečeká na zaplnění celé palety.
- Po nandání bedniček z regálového skladu na podvozek, zakladač položí podvozek na zem (viz Obr. 5-5 - bod **8**) → naložené vagóny čekají na vláček → po příjezdu vláčku se provede zapřažení všech naložených vagónů k vláčku → načtení čárového kódu → vláček odveze vagón k lince, kde vagón zanechá a pokračuje k další lince. Postupně odpojí všechny vagóny.

Popis chování skladu a dopravy pro malé dílce – doprava prázdných beden, vrácenek a odpadu

- Pokud vláček veze vagóny s vrácenkami od montážních linek, nebo veze odpad od montážních linek, nebo veze hotové výrobky z předvýroby, pak tyto vagóny se odpojí v bodu **10** (viz Obr. 5-5).
- Vláček s vagóny, na kterých se nachází prázdné bedničky, pokračuje do bodu **11** (viz Obr. 5-5) → odpojení všech vagónů → v bodu **12** (viz Obr. 5-5) se k vláčku připojí tolik prázdných vagónků, kolik se jich odpojilo v bodu **11**.

- Vláček jede s prázdnými vagóny do bodu 8 (viz Obr. 5-5). Zde se odpojí prázdné vagóny.

5.5.1.2 **Doprava velkých dílců pro montážní linky**

Popis chování skladu a dopravy pro velké dílce – naložení:

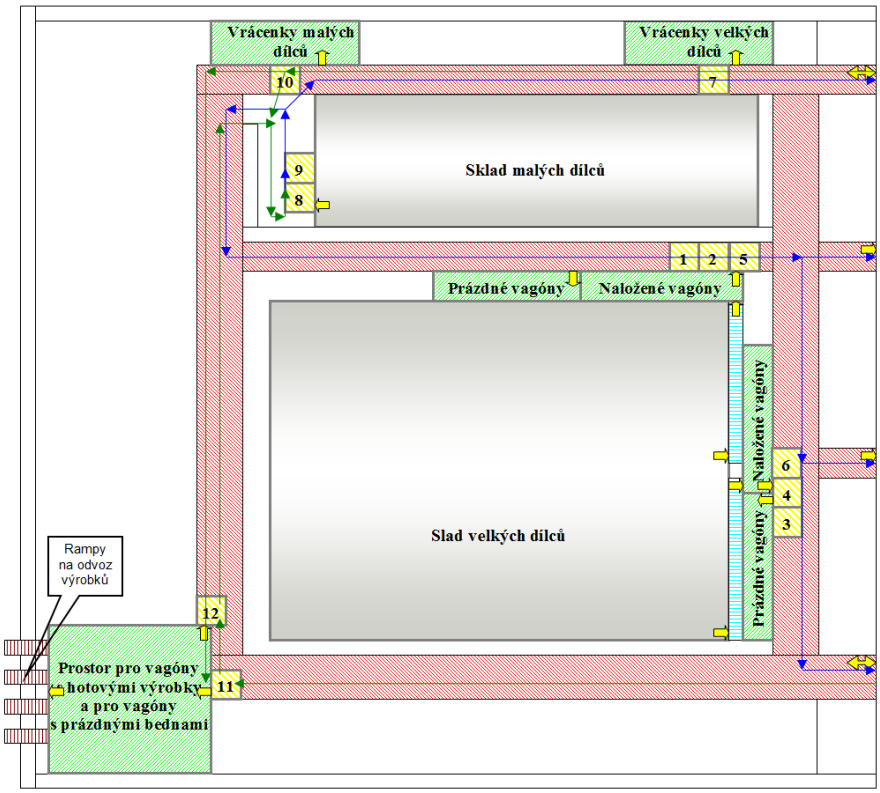
- Vyskladnění z regálového skladu je modelováno pomocí kapacity zakladače. Regálový zakladač položí bednu na konec válečkového dopravníku (každý válečkový dopravník má 50% pravděpodobnost, že na něj regálový zakladač položí bednu). Na konci každé válečkové trati je pracovník, který zajistí položení bedny z válečkového dopravníku na vagón – bod **2**, **4** (viz Obr. 5-6) → naložené vagóny čekají na vláček v prostoru určené pro naložené vagóny → provede se zapřažení všech naložených vagónů k vláčku - bod **2**, **4** (vláček může jet i s jedním vagónem) → načtení čárového kódu - bod **5**, **6** (viz Obr. 5-6) → vláček odveze vagón k lince, kde vagón zanechá v nádraží a pokračuje k další lince. Postupně odpojí všechny vagóny.

Popis chování skladu a dopravy pro velké dílce – doprava prázdných beden a vrácenek

- Pokud vláček veze vagóny s vrácenkami od montážních linek, pak se tyto vagóny odpojí v bodu **7** (viz Obr. 5-6) → vláček s vagóny, na kterých se nachází prázdné bedny, pokračuje do bodu **11**, kde se odpojí všechny vagóny → v bodu **12** se k vláčku připojí tolik prázdných vagónků, kolik se jich odpojilo v bodu **11** → vláček jede s prázdnými vagóny do bodu **1**, nebo **3**, kde odpojí prázdné vagóny. Vláčky dodržují svoje okruhy. Tzn., pokud vyjede z bodu **2**, tak se do něj musí opět vrátit (nemůže střídat výjezdy naložených vagónů z bodu **2**, nebo z bodu **4**)
- Vláček v bodu **2**, nebo **4** připojí jednotlivé vagóny.

5.5.1.3 **Expedice hotových výrobků od montážních linek.**

- Hotové výrobky z montážních linek se přivezou do skladu k rampám, kde čekají na odvoz nákladním autem.
- Hotové výrobky jsou nakládány přes rampu.
- Odvoz odpadů je uskutečněn přes rampu.



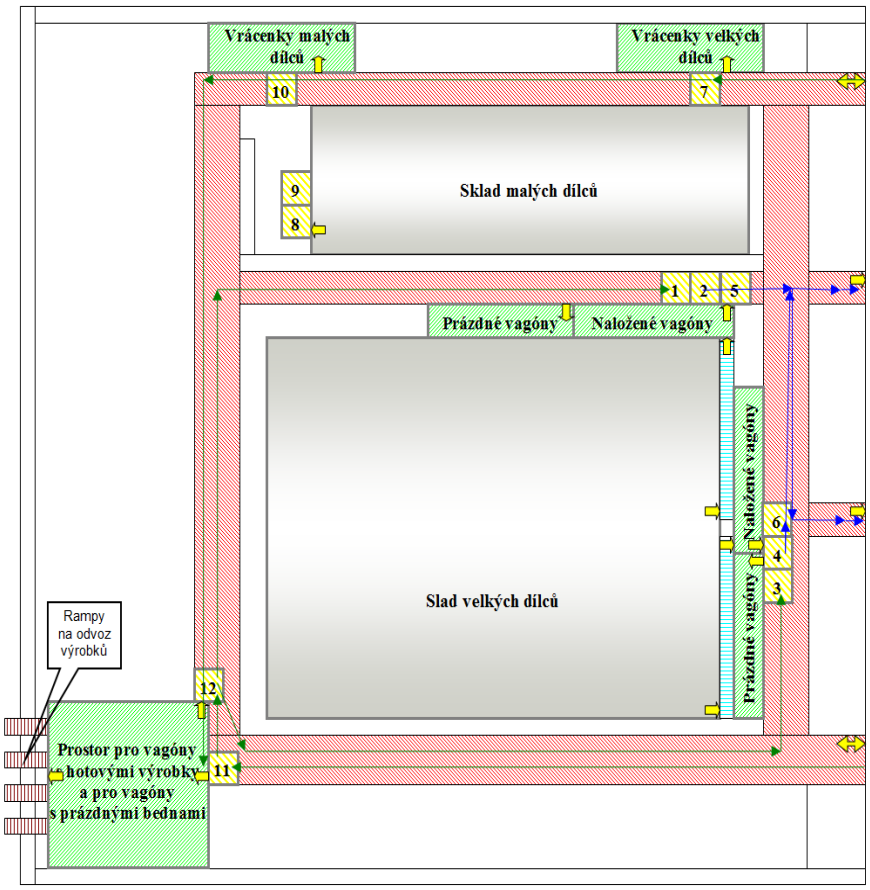
Legenda:

1. Vyložení prázdných vagónů pro velké dílce – 1. výstup.
2. Naložení vagónu s bednou velkých dílců – 1. výstup.
3. Vyložení prázdných vagónů pro velké dílce – 2. výstup.
4. Naložení vagónu s bednou velkých dílců – 2. výstup.
5. Načtení čárového kódu velkého dílce.
6. Načtení čárového kódu velkého dílce.
7. Místo odpojení vagónu vrácenek velkých dílců.
8. Vyložení prázdných vagónů pro malé dílce.
9. Načtení čárového kódu malých dílců.
10. Místo odpojení vagónu vrácenek malých dílců.
11. Odpojení vagónu s hotovými výrobky a prázdnými bednami.
12. Připojení prázdných vagónů.

→ … Cesty odvozu malých dílců ze skladu do výroby.

→ … Cesty prázdných vagónů malých dílců z výroby do skladu.

Obr. 5-5 Sklad – doprava malých dílců



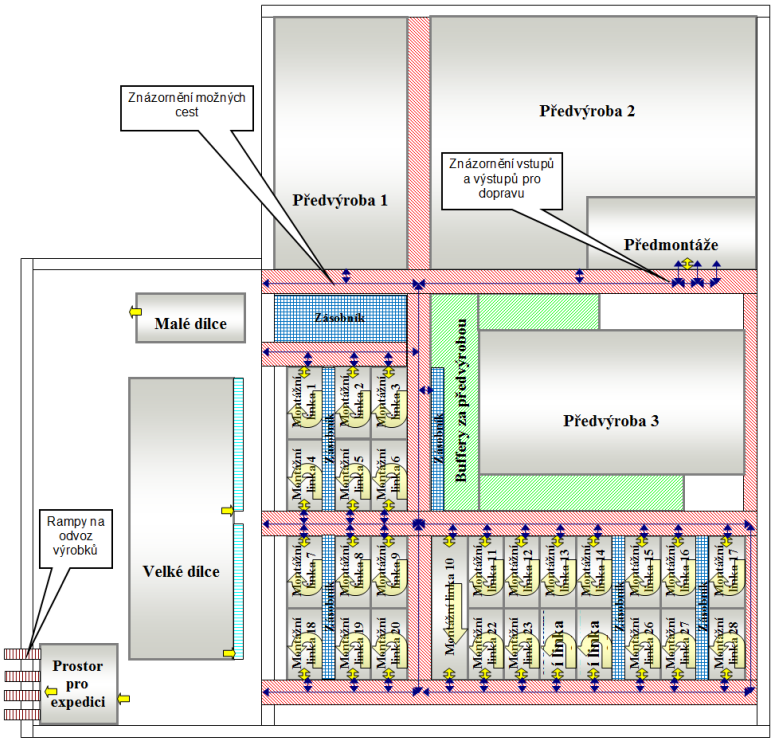
Legenda:

1. Vyložení prázdných vagónů pro velké dílce – 1. výstup.
2. Naložení vagónu s bednou velkých dílců – 1. výstup.
3. Vyložení prázdných vagónů pro velké dílce – 2. výstup.
4. Naložení vagónu s bednou velkých dílců – 2. výstup.
5. Načtení čárového kódu velkého dílce.
6. Načtení čárového kódu velkého dílce.
7. Místo odpojení vagónu vrácenek velkých dílců.
8. Vyložení prázdných vagónů pro malé dílce.
9. Načtení čárového kódu malých dílců.
10. Místo odpojení vagónu vrácenek malých dílců.
11. Odpojení vagónu s hotovými výrobky a prázdnými bednami.
12. Připojení prázdných vagónů.

… Cesty odvozu malých dílců ze skladu do výroby.

… Cesty prázdných vagónů malých dílců z výroby do skladu.

Obr. 5-6 Sklad – doprava velkých dílců



Obr. 5-7 Schéma modelované výrobní haly

5.5.2 Rozhodovací proměnné

Jako rozhodovací proměnné jsou nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	Počet vláčků pro odvážení malých dílců pro montážní linky a pro předmontáže v předvýrobě
	X_2	Počet vláčků pro odvážení velkých dílců pro montážní linky
	X_3	Počet vláčků pro odvážení hotových výrobků od montážních linek

Tabulka 5-25 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	8671	X_1	3	15	1	7
		X_2	7	35	1	5
		X_3	3	25	1	5

Tabulka 5-26 Specifikace prohledávaného prostoru

5.5.3 Účelová funkce

Účelová funkce odráží průměrné využití vláčků s vagóny, které slouží pro transport malých a velkých dílců a hotových výrobků a celkové průměrné využití všech montážních linek. Průměrné využití všech montážních linek je v účelové funkci nadřazeno průměrnému využití u všech typů vláčků pomocí koeficientů. Jedná se o maximalizaci účelové funkce.

Účelová funkce	Popis	Minimalizace výsledné účelové funkce
$F(\mathbf{X})$	Výsledná účelová funkce	Ne
$F_1(\mathbf{X})$	Suma průměrného využití všech linek	
$F_2(\mathbf{X})$	Průměrné využití vláčků pro malé dílce	
$F_3(\mathbf{X})$	Průměrné využití vláčků pro velké dílce	
$F_4(\mathbf{X})$	Průměrné využití vláčků pro odvoz hotových výrobků	

Tabulka 5-27 Specifikace účelové funkce

Předpis účelové funkce:

$$F_1(\mathbf{X}) = \sum_{i=1}^{28} \varphi_i \tag{5.5}$$

kde:

- φ_i … Průměrné využití *i*-té montážní linky.

$$F(\mathbf{X}) = F_1(\mathbf{X}) + \frac{F_2(\mathbf{X})}{3} + \frac{F_3(\mathbf{X})}{3} + \frac{F_4(\mathbf{X})}{3} \tag{5.6}$$

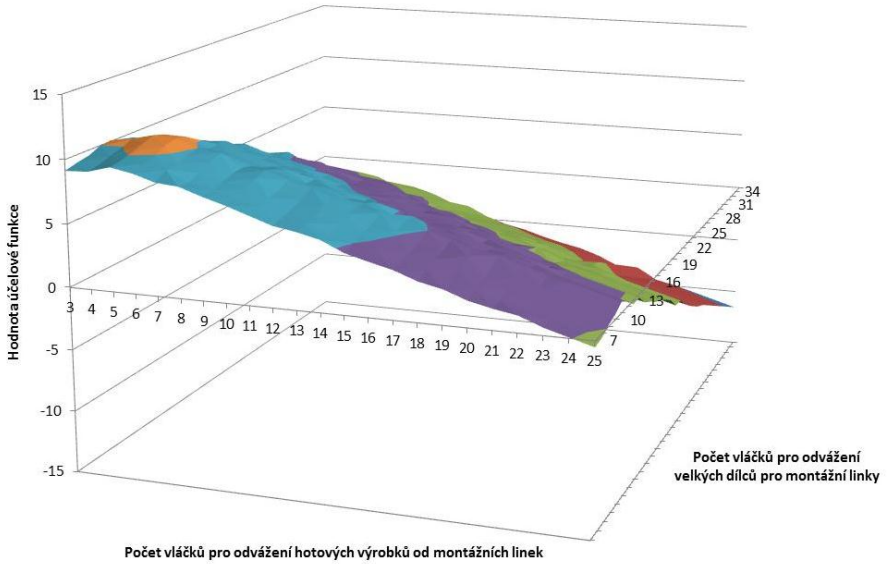
kde:

- $F(\mathbf{X})$ … Výsledná účelová funkce.

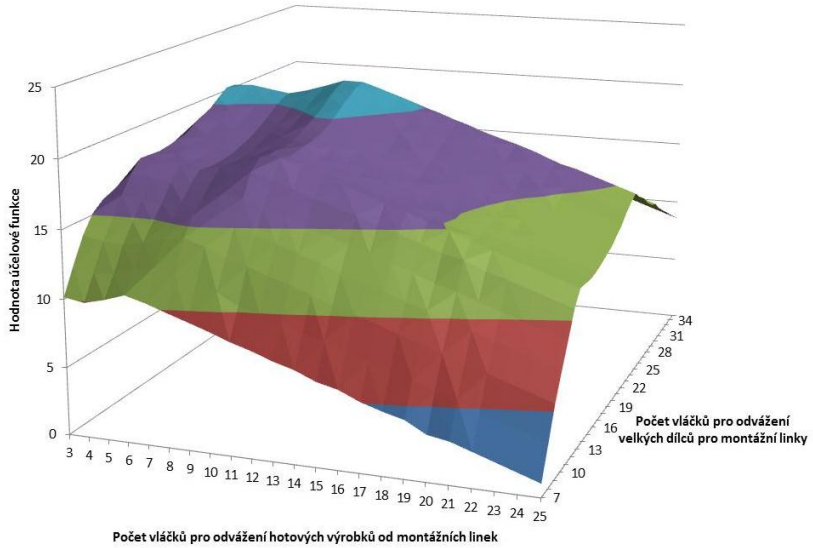
Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	3	-11.53367	11	22.64956
		X_2	35		24	
		X_3	25		11	

Tabulka 5-28 Globální minimum a maximum účelové funkce v prohledávaném prostoru

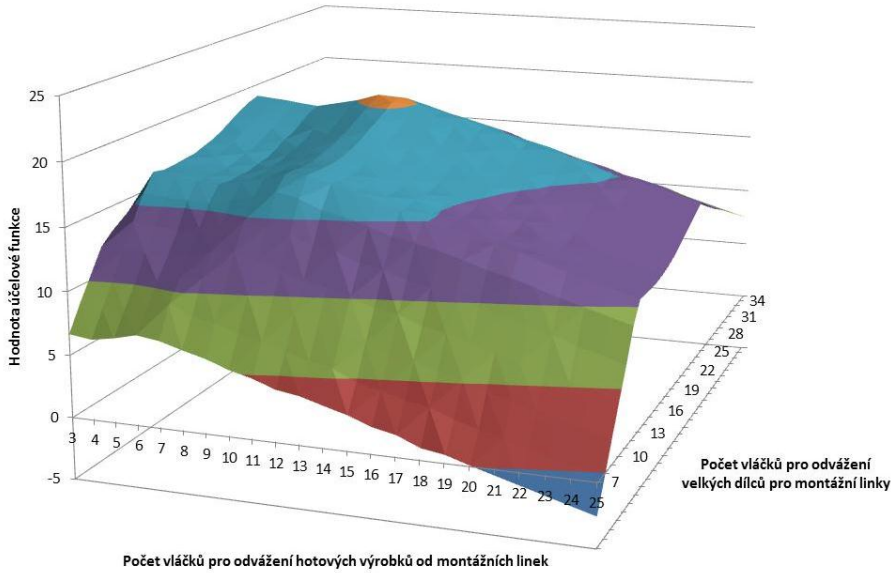
Protože účelová funkce je závislá na třech rozhodovacích proměnných, jsou v následujících obrázcích vykresleny tři průběhy účelové funkce při použití filtru na hodnoty první rozhodovací proměnné (Počet vláčků pro odvážení malých dílců pro montážní linky a pro předmontáže v předvýrobě) z důvodu nejmenšího rozsahu (počtu) hodnot na ose ze všech rozhodovacích proměnných.



Obr. 5-8 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 3;



Obr. 5-9 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 8;



Obr. 5-10 Průběh účelové funkce modelu dopravy při nastavení filtru: Vláček malé bedny = 15;

5.5.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	4000	22.64956	0.001	
Downhill Simplex, Diferenciální evoluce	500	22.64956	0.001	0.003

Tabulka 5-29 Specifikovaná kritéria ukončení

5.5.5 Simulační experimenty

Parametry běhu na simulačním modelu jsou nastaveny na tyto hodnoty:

Parametry experimentu na simulačním modelu	Hodnota	Časová jednotka
Simulační doba v simulačním modelu	4	Dny
Doba náběhu simulačního modelu	2	Dny
Pracovní doba	24	Hod/Den

Tabulka 5-30 Parametry běhu na simulačním modelu

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu (t[sec])
Doprava	208.8

Tabulka 5-31 Čas potřebný k proběhnutí simulačního experimentu

5.6 Simulační model „Penalizace“

5.6.1 Popis modelu

Simulační model představuje výrobní dílnu, kde jsou vyráběny dva druhy výrobků. Výrobní dílna obsahuje 8 technologicky orientovaných pracovišť. Každé pracoviště disponuje stanoveným počtem zdrojů. Tento počet byl určen na základě statického odhadu počtu zdrojů podle stanoveného objemu produkce a definovaného času obrábění na pracovišti.

Každý výrobek má svůj technologický postup – čas obrábění a operace, která má být provedena na daném typu výrobku. Cílem je vyrobit 100 kusů prvního výrobku a 200 kusů druhého výrobku za dobu 6350 minut. Za nevyrobení prvního výrobku je stanovena penalizace 200 Kč a za druhý je penalizace stanovena částkou 300 Kč. Pokud výrobek přesáhne stanovenou průběžnou dobu výrobu, bude penalizován. Penalizace nastává i v případě, že výrobek je předčasně vyroben. V rámci simulačního modelu budou měněny vstupy obou výrobků na dílnu.

Jedná se o minimalizaci účelové funkce, která odráží výši nákladů penalizace. Globální minimum účelové funkce v dvourozměrném prohledávaném prostoru je $F(\tilde{\mathbf{X}}) = F(3004.0; 1796.0) \approx 100.71, \tilde{\mathbf{X}} \in \tilde{X}$.

5.6.2 Rozhodovací proměnné

Jako rozhodovací proměnné nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	Čas vstupu prvního výrobku na výrobní dílnu
	X_2	Čas vstupu druhého výrobku na výrobní dílnu

Tabulka 5-32 Specifikace rozhodovacích proměnných v prohledávaném prostoru

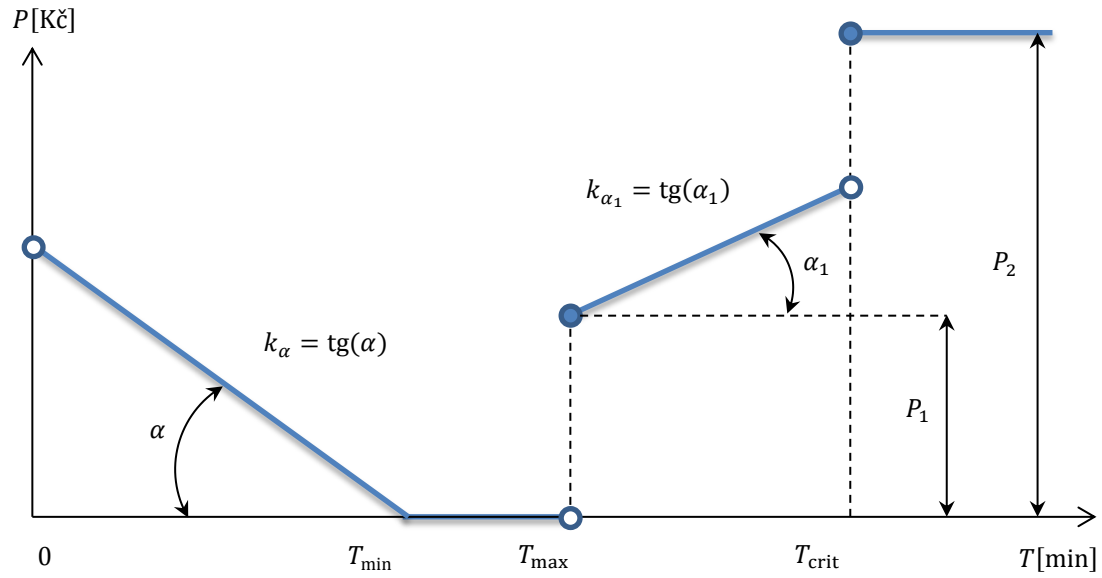
Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	53001	X_1	2900	3050	1	2900
		X_2	1500	1850	1	1500

Tabulka 5-33 Specifikace prohledávaného prostoru

5.6.3 Účelová funkce

Účelová funkce vyjadřuje míru nesplnění stanovené časové lhůty pro dodání obou typů výrobků. Následující obrázek (viz Obr. 5-11) vyjadřuje míru penalizace za nedodržení stanovené průběžné doby výroby.



Obr. 5-11 Podmínky penalizace - nedodržení stanovené průběžné doby výroby [44]

$P \leftarrow \mathrm{Penalty}(T, T_{\min}, T_{\max}, T_{\mathrm{crit}}, k_\alpha, k_{\alpha_1}, P_1, P_2)$	
Funkce, jejímž výstupem je míra penalizace za nesplnění stanovené průběžné doby výroby.	
Vstup:	T: Průběžná doba výroby
Vstup:	$T_{\min}$: Minimální požadovaná hodnota průběžné doby výroby
Vstup:	$T_{\max}$: Maximální požadovaná hodnota průběžné doby výroby
Vstup:	T_{crit}: Kritická hodnota průběžné doby výroby
Vstup:	k_α: Konstanta vyjadřující směrnici přímky penalizace za předčasnou výrobu
Vstup:	k_{α_1}: Konstanta vyjadřující směrnici přímky penalizace překročení stanovené průběžné doby výroby
Vstup:	P_1: Konstanta vyjadřující výši penalizace za překročení stanovené průběžné doby výroby
Vstup:	P_2: Konstanta vyjadřující výši penalizace za překročení stanovené kritické průběžné doby výroby
Výstup:	P: Hodnota penalizace výrobku

```
11  begin
12    if (T > 0) and (T < T_min) then    //penalizace za předčasnou výrobu
13        P ← (T_min − T) * k_α;
14    if (T ≥ T_min) and (T < T_max) then    //žádná penalizace
15        P ← 0;
16    if (T = T_max) then
      //stanovená penalizace za překročení maximální požadované doby
17        P ← P_1;
18    if (T > T_max) and (T < T_crit) then
      //nárůst penalizace v závislosti na překročení maximální požadované doby
19        P ← P_1 + (T − T_max) * k_α1;
20    else        // maximální možná penalizace
21        P ← P_2;
```

```
22        result ⟵ P;
23      end;
```

Algoritmus 5-1 Výše penalizace u výrobku

Hodnota $T_{\max}$ neodpovídá čistému času výroby jednoho výrobku - tedy hodnotám 200 a 340 minut. Tento čistý čas byl upraven koeficientem k = 1,05 tak, abychom se více přiblížili v praxi dosažitelným a ve skutečném výrobním systému realizovatelným hodnotám. [44]

Účelová funkce	Popis	Minimalizace výsledné účelové funkce
$F(\mathbf{X})$	Výsledná účelová funkce - výsledná výše penalizace	Ano
$F_1(\mathbf{X})$	Účelová funkce - penalizace za překročení stanovené průběžné doby výroby	
$F_2(\mathbf{X})$	Účelová funkce - penalizace za nedodělky	

Tabulka 5-34 Specifikace účelové funkce

Předpis účelové funkce:

$$F(\mathbf{X}) = F_1(\mathbf{X}) + F_2(\mathbf{X}) \tag{5.7}$$

$$F_1(\mathbf{X}) = \sum_{i=1}^{q} P_i \tag{5.8}$$

$$F_2(\mathbf{X}) = \sum_{i=1}^{q_{\text{Total}}-q} Pn_i \tag{5.9}$$

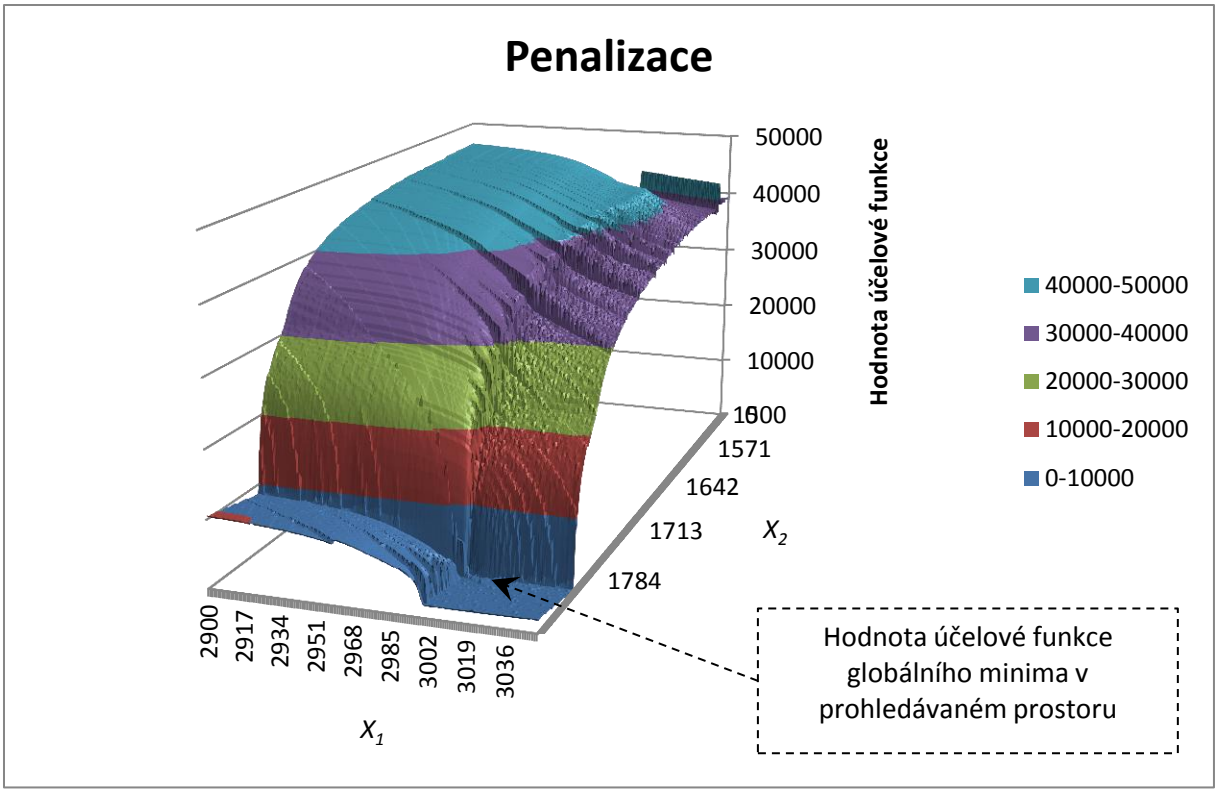
kde:

- q …Počet vyrobených výrobků.
- P_i … Penalizace za překročení stanovené průběžné doby výroby.
- q_{Total} … Stanovený počet výrobků, které mají být vyrobeny.
- Pn … Penalizace za nevyrobení jednoho kusu výrobku.

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	3004	100.7092832	2900	46426.34066
		X_2	1796		1500	

Tabulka 5-35 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-12) je zobrazen průběh účelové funkce simulačního modelu „Penalizace“. Hodnota účelové funkce globálního minima je umístěna v dolním pravém rohu v rovinné oblasti.



Obr. 5-12 Průběh účelové funkce modelu penalizace

5.6.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	1000	100.7092832	0.001	
Downhill Simplex, Diferenciální evoluce	500	100.7092832	0.001	0.003

Tabulka 5-36 Specifikovaná kritéria ukončení

5.6.5 Simulační experimenty

Parametry běhu na simulačním modelu jsou nastaveny na tyto hodnoty:

Parametry experimentu na simulačním modelu	Hodnota	Časová jednotka
Simulační doba v simulačním modelu	6350	Min
Doba náběhu simulačního modelu	0	Dny
Pracovní doba	24	Hod/Den

Tabulka 5-37 Parametry běhu na simulačním modelu

Čas potřebný k proběhnutí simulačního experimentu:

Simulační model	Čas potřebný pro proběhnutí simulačního experimentu (t[sec])
Penalizace	2.7

Tabulka 5-38 Čas potřebný k proběhnutí simulačního experimentu

5.7 Simulační model „VyrobniLinka“

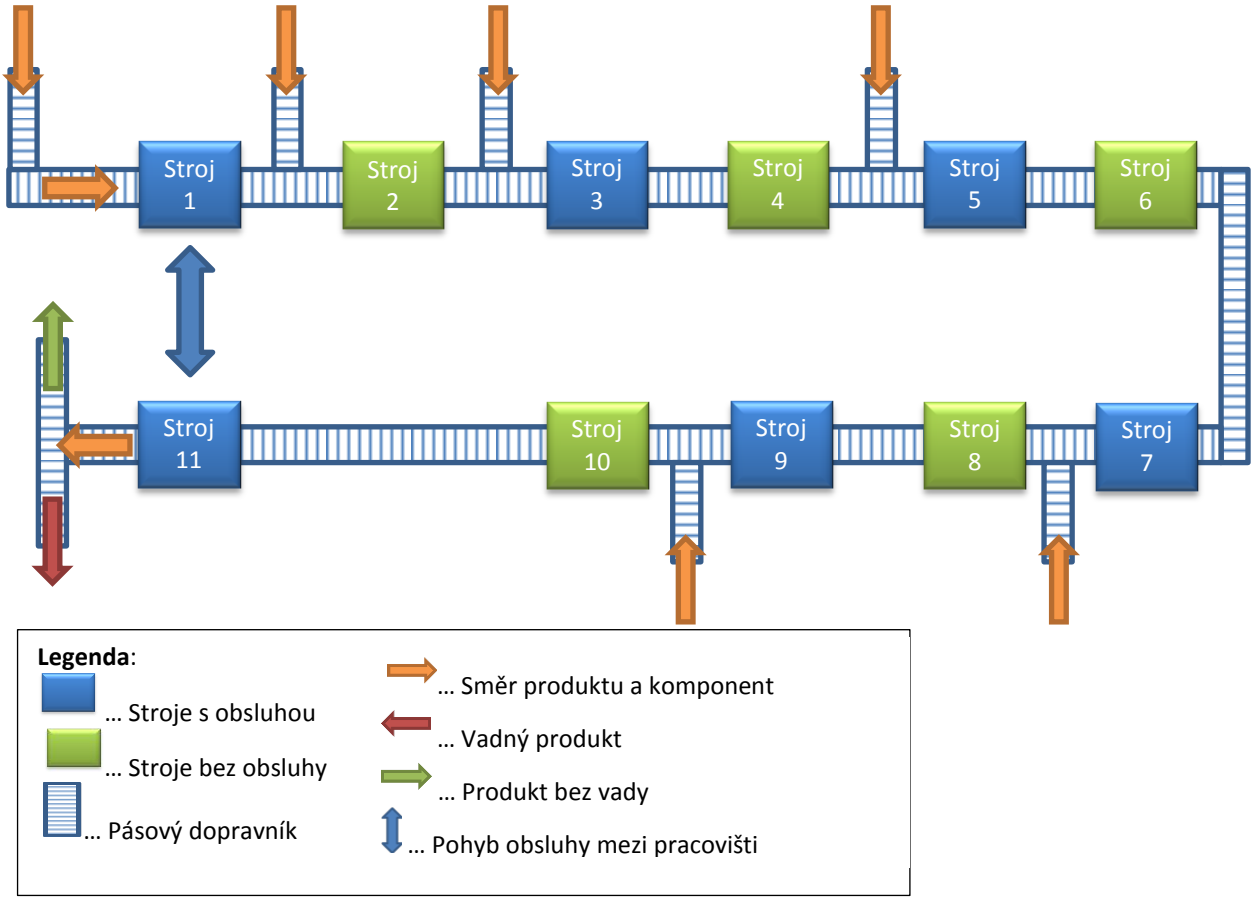
5.7.1 Popis simulačního modelu

Simulační model charakterizuje výrobní linku pro dva různé výrobní postupy včetně relativní chybovosti na jednotlivých pracovištích. Doprava ve výrobě je uskutečněna pomocí pásového dopravníku. Linka se skládá z 11 pracovišť (viz Obr. 5-13). Jsou definovány jednotlivé časy zpracování a jejich odchylky. Na konci výrobní linky jsou výrobky tříděny podle toho, zda jsou vadné či nikoliv. Pokud vznikl na jakémkoliv pracovišti zmetek, tento zmetek již na následujících pracovištích není dále zpracován. Je také stanovena celková doba simulace, která činí 14 dnů ve třísměnném provozu.

Rozhodovací proměnné jsou:

- Počet upínacích přípravků v oběhu.
- Počet čekajících přípravků na pásu inicializující přechod pracovníka mezi pracovišti se strojem 1 a strojem 11.

Jedná se o maximalizace účelové funkce. Globální maximum účelové funkce v dvourozměrném prohledávaném prostoru je $F(\hat{\mathbf{X}}) = F(18.0,4.0) = 22032, \hat{\mathbf{X}} \in \tilde{X}$.



Obr. 5-13 Schéma výrobní linky

5.7.2 Rozhodovací proměnné – vstupní parametry simulačního modelu

Jako rozhodovací proměnné nastaveny:

Prohledávaný prostor	Rozhodovací proměnná - osa (X_j)	Popis
$\tilde{X}$	X_1	Počet upínacích přípravků ve výrobě
	X_2	Počet upínacích přípravků inicializující přechod pracovníka mezi pracovišti se strojem 1 a strojem 11

Tabulka 5-39 Specifikace rozhodovacích proměnných v prohledávaném prostoru

Prohledávaný prostor:

Prohled. prostor	Počet kandidátů řešení – prvků - v prohled. prostoru	Rozhodovací proměnná - osa (X_j)	Dolní mez na ose (a_j)	Horní mez na ose (b_j)	Krok na ose $(x_i - x_{i-1})$	Souřadnice počátečního přípustného řešení
$\tilde{X}$	238	X_1	7	40	1	7
		X_2	1	7	1	4

Tabulka 5-40 Specifikace prohledávaného prostoru

5.7.3 Účelová funkce

Účelová funkce odráží počet vyrobených produktů bez vady na výrobní lince. Funkce je penalizována počtem upínacích přípravků, které jsou v oběhu u výrobní linky. Snahou je maximalizovat hodnotu výsledné účelové funkce.

Účelová funkce	Popis	Minimalizace výsledné účelové funkce
$F(\mathbf{X})$	Výsledná účelová funkce	Ne
$F_1(\mathbf{X})$	Počet produktů bez vady	
$F_2(\mathbf{X})$	Počet upínacích přípravků ve výrobě	

Tabulka 5-41 Specifikace účelové funkce

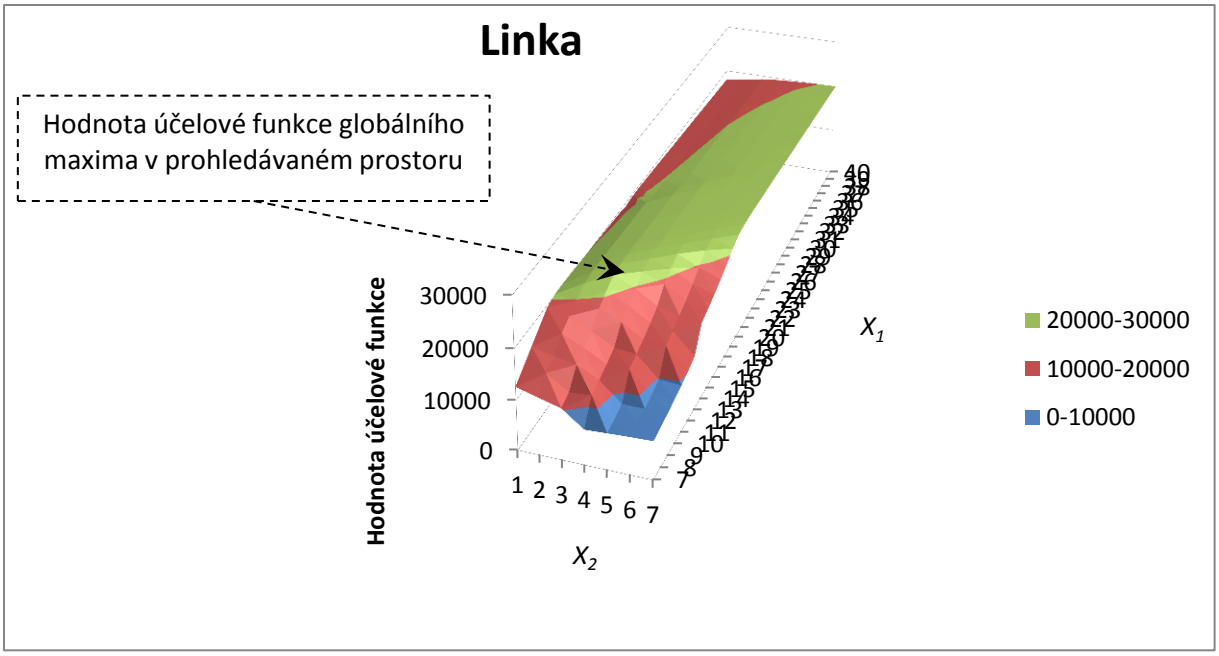
Předpis účelové funkce:

$$F(\mathbf{X}) = F_1(\mathbf{X}) - F_2(\mathbf{X}) \cdot 20 \tag{5.10}$$

Účelová funkce	Prohled. prostor	Rozhodovací proměnná - osa (X_j)	Globální minimum $(\check{\mathbf{X}})$ účelové funkce v prohledávaném prostoru		Globální maximum $(\hat{\mathbf{X}})$ účelové funkce v prohledávaném prostoru	
			Hodnota rozhod. proměnné	Hodnota účelové funkce	Hodnota rozhod. proměnné	Hodnota účelové funkce
$F(\mathbf{X})$	$\tilde{X}$	X_1	7	7092	18	22032
		X_2	4		4	

Tabulka 5-42 Globální minimum a maximum účelové funkce v prohledávaném prostoru

V následujícím grafu (viz Obr. 5-14) je zobrazen průběh účelové funkce v prohledávaném prostoru. Nejvyšší hodnota účelové funkce globální maxima je na vrcholu kopce.



Obr. 5-14 - Průběh účelové funkce modelu výrobní linky

5.7.4 Kritérium ukončení

U vybraného modelu jsou specifikovány následující kritéria ukončení:

Optimalizační metoda	Způsob ukončení optimalizace			
	Ukončit po definovaném počtu iterací	Ukončit při dosažené hodnotě		Poměr zlepšení sub-optima
		Specifická hodnota	Delta	
Random Search, Hill Climbing, Tabu Search, Simulated Annealing, Local Search, Evoluční Strategie	238	22032	0.001	
Downhill Simplex, Diferenciální evoluce	238	22032	0.001	0.003

Tabulka 5-43 Specifikovaná kritéria ukončení

5.7.5 Simulační experimenty

Parametry běhu na simulačním modelu jsou nastaveny na tyto hodnoty:

Parametry experimentu na simulačním modelu	Hodnota	Časová jednotka
Simulační doba v simulačním modelu	14	Dny
Doba náběhu simulačního modelu	0	Dny
Pracovní doba	24	Hod/Den

Tabulka 5-44 Parametry běhu na simulačním modelu

Čas potřebný k proběhnutí simulačního experimentu:

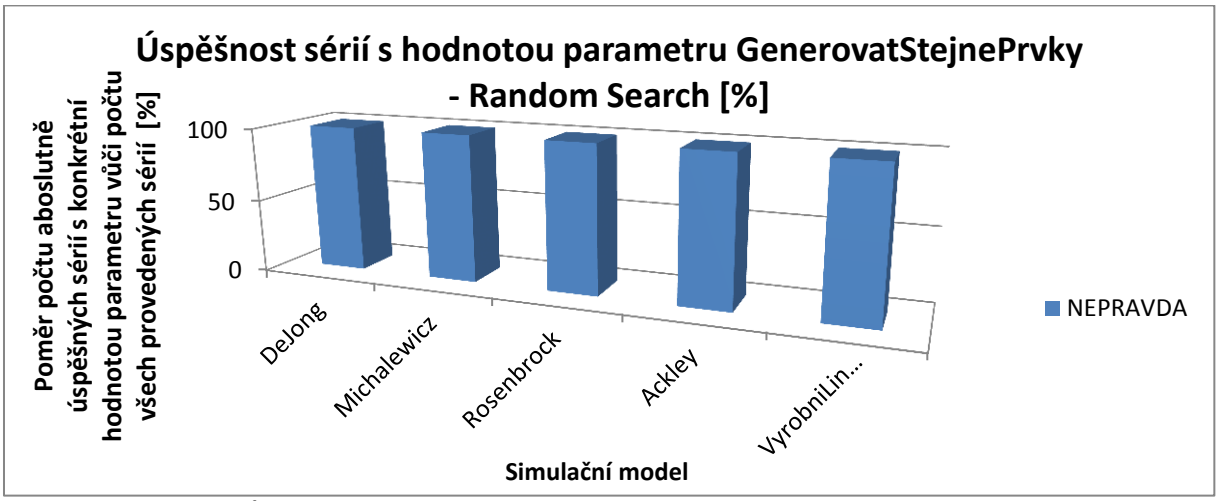
Simulační model	Čas potřebný pro proběhnutí simulačního experimentu (t[sec])
Linka	6.42

Tabulka 5-45 Čas potřebný k proběhnutí simulačního experimentu

6 Vhodné nastavení parametrů jednotlivých optimalizačních algoritmů na základě provedených sérií

6.1 Random Search

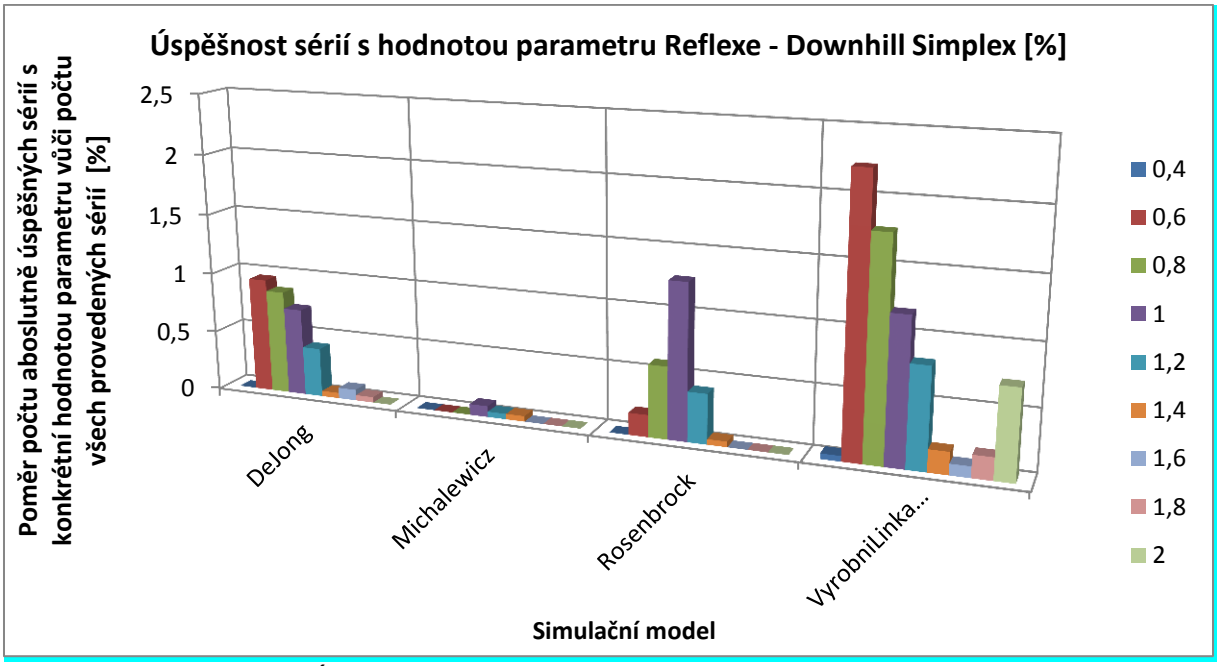
6.1.1 Generovat stejné prvky



Obr. 6-1 Úspěšnost sérií s hodnotou parametru „Generovat stejné prvky“ - Random Search

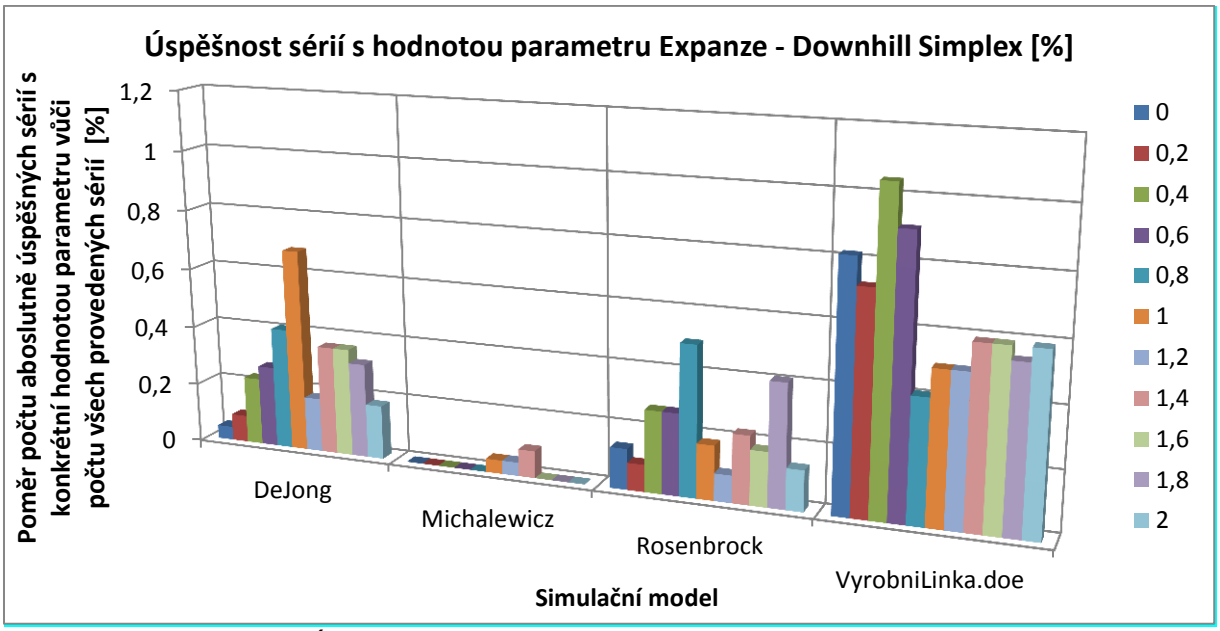
6.2 Downhill Simplex

6.2.1 Reflexe



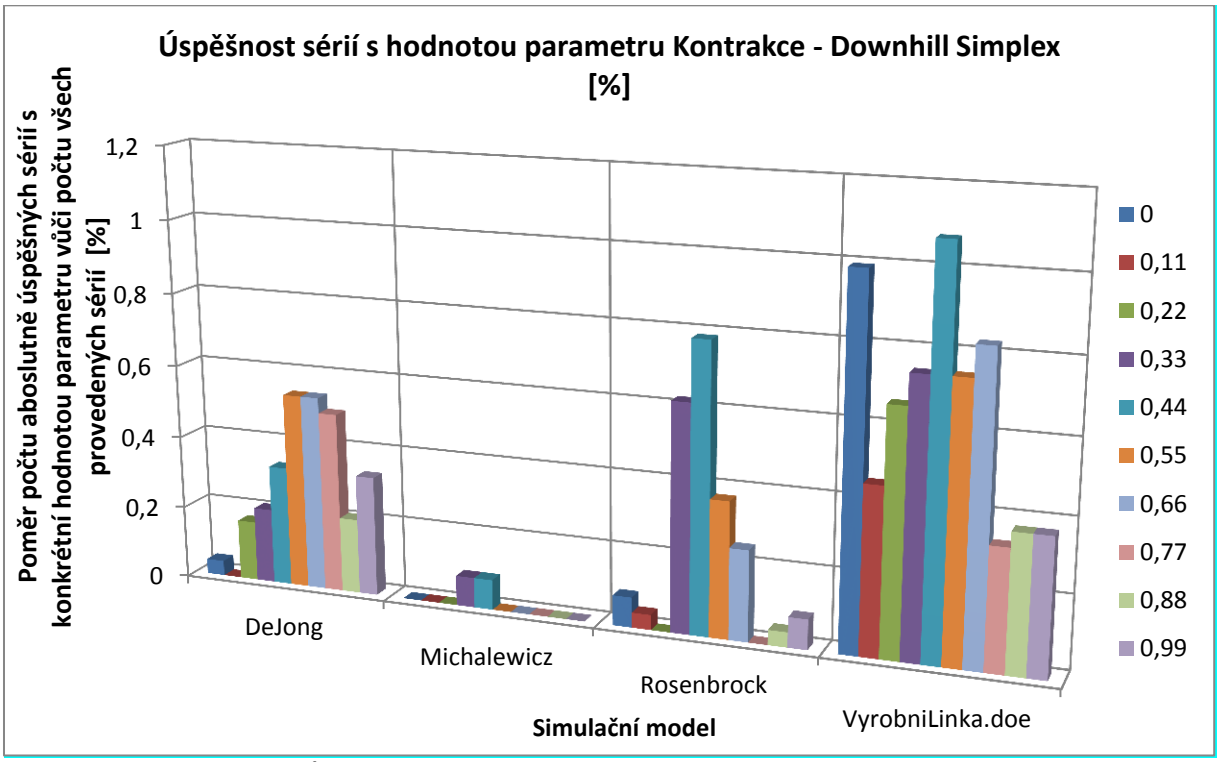
Obr. 6-2 Úspěšnost sérií s hodnotou parametru „Reflexe“ – Downhill Simplex

6.2.2 Expanze



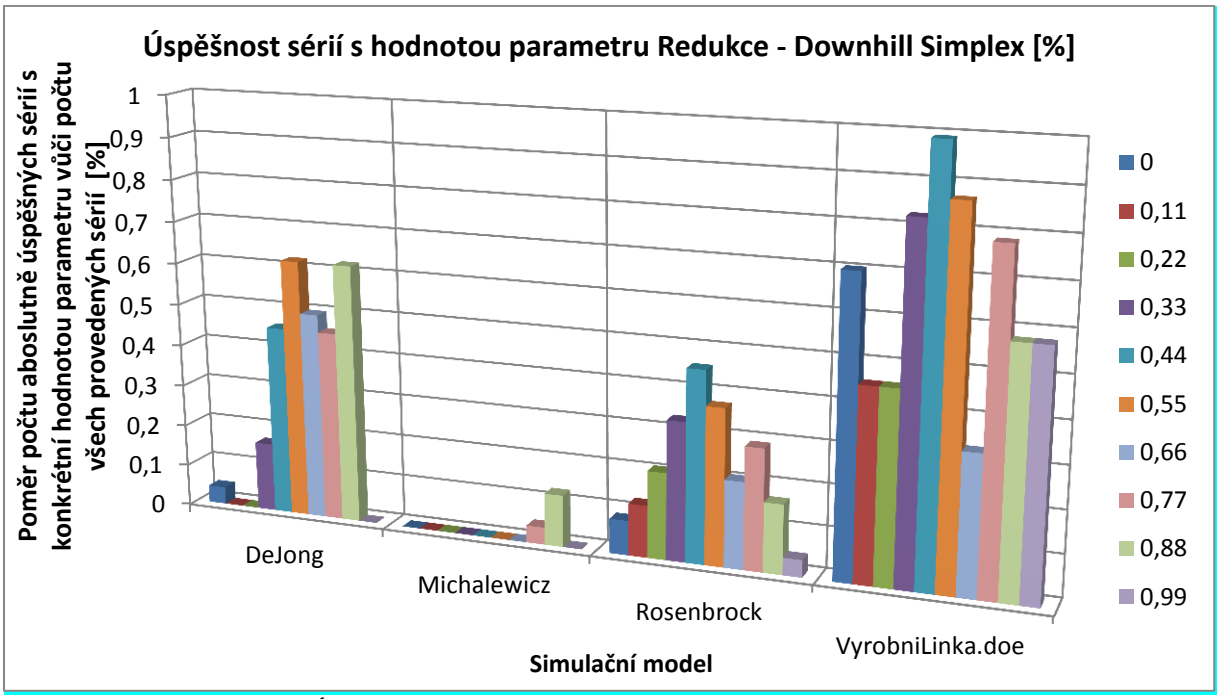
Obr. 6-3 Úspěšnost sérií s hodnotou parametru „Expanze“ - Downhill Simplex

6.2.3 Kontrakce



Obr. 6-4 Úspěšnost sérií s hodnotou parametru „Kontrakce“- Downhill Simplex

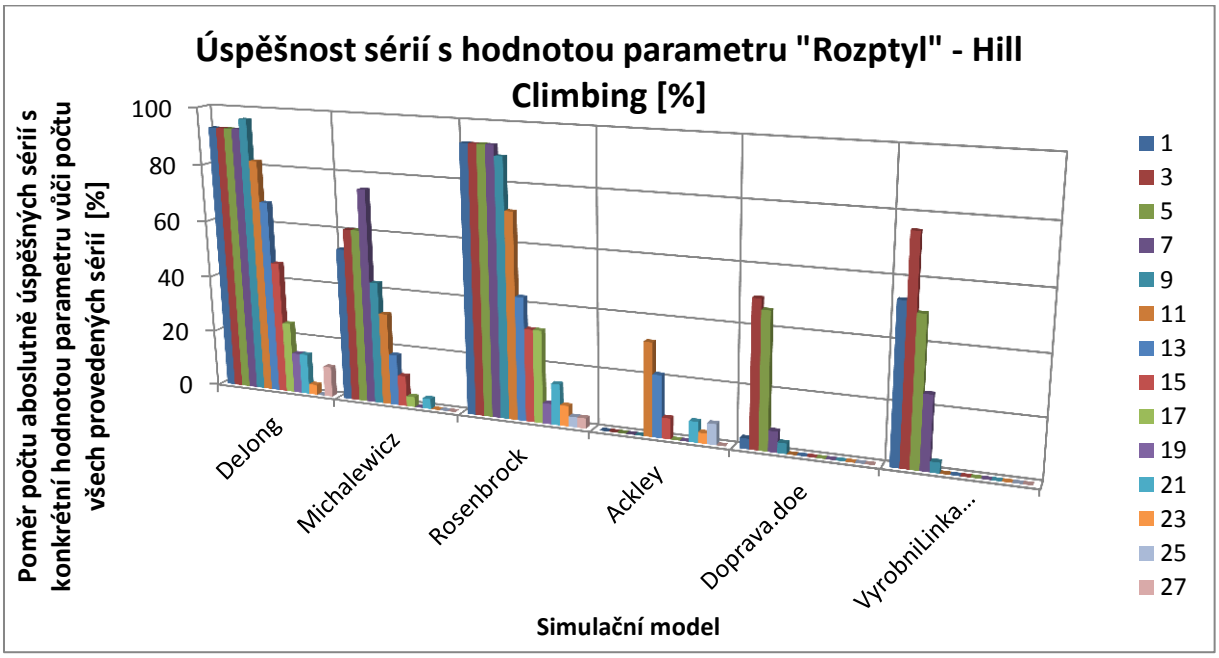
6.2.4 Redukce



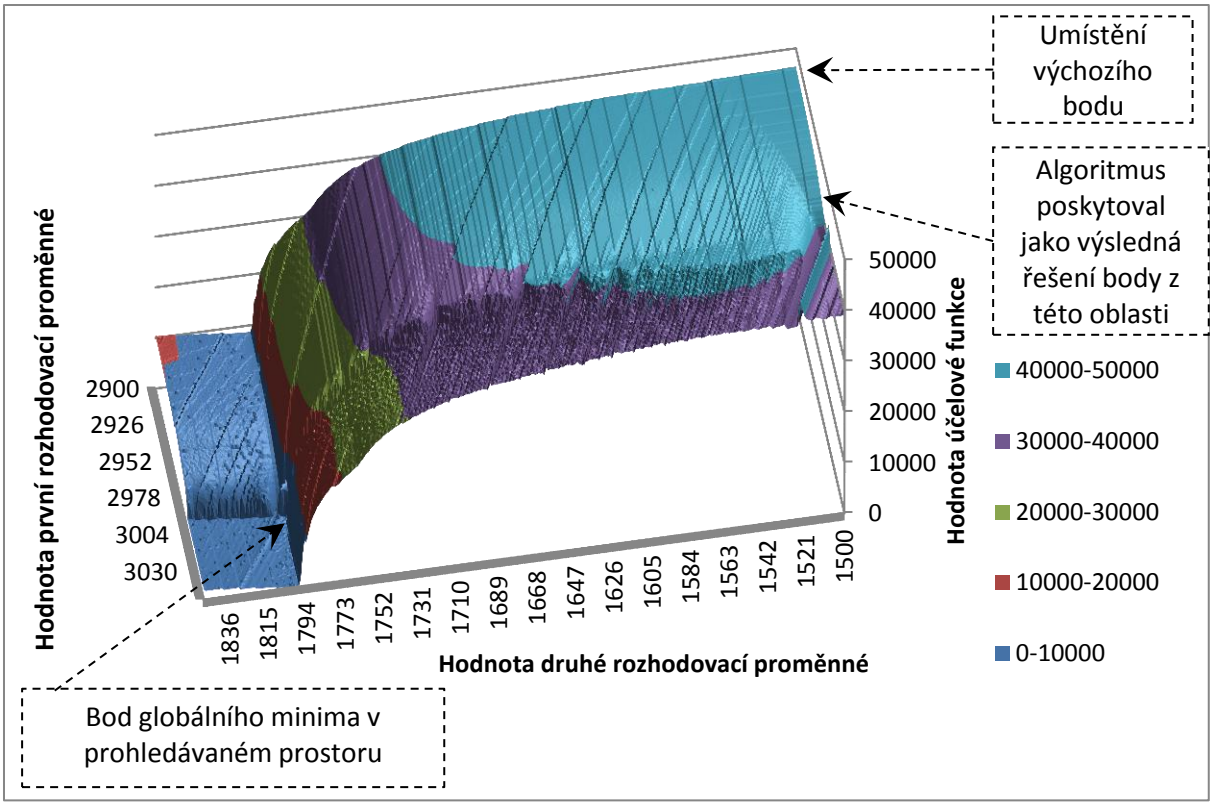
Obr. 6-5 Úspěšnost sérií s hodnotou parametru „Redukce“- Downhill Simplex

6.3 Hill Climbing

6.3.1 Rozptyl

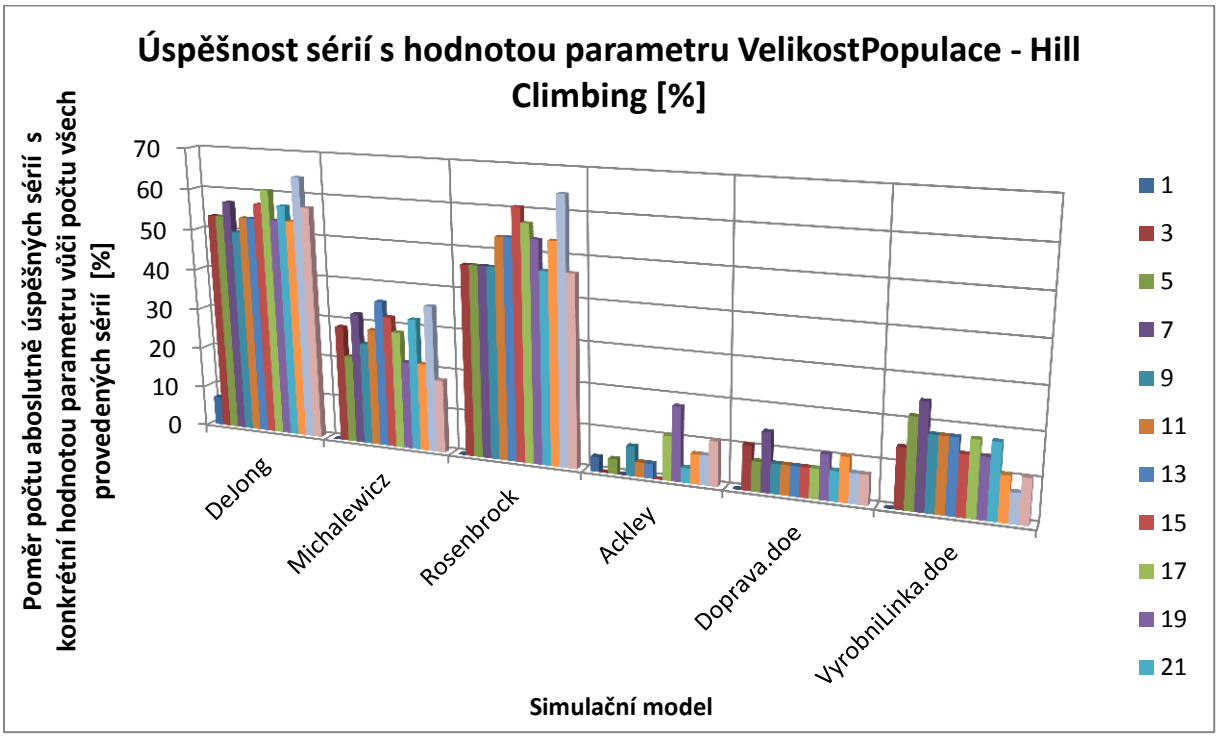


Obr. 6-6 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Hill Climbing



Obr. 6-7 Horní pohled na povrch účelové funkce simulačního modelu „Penalizace“

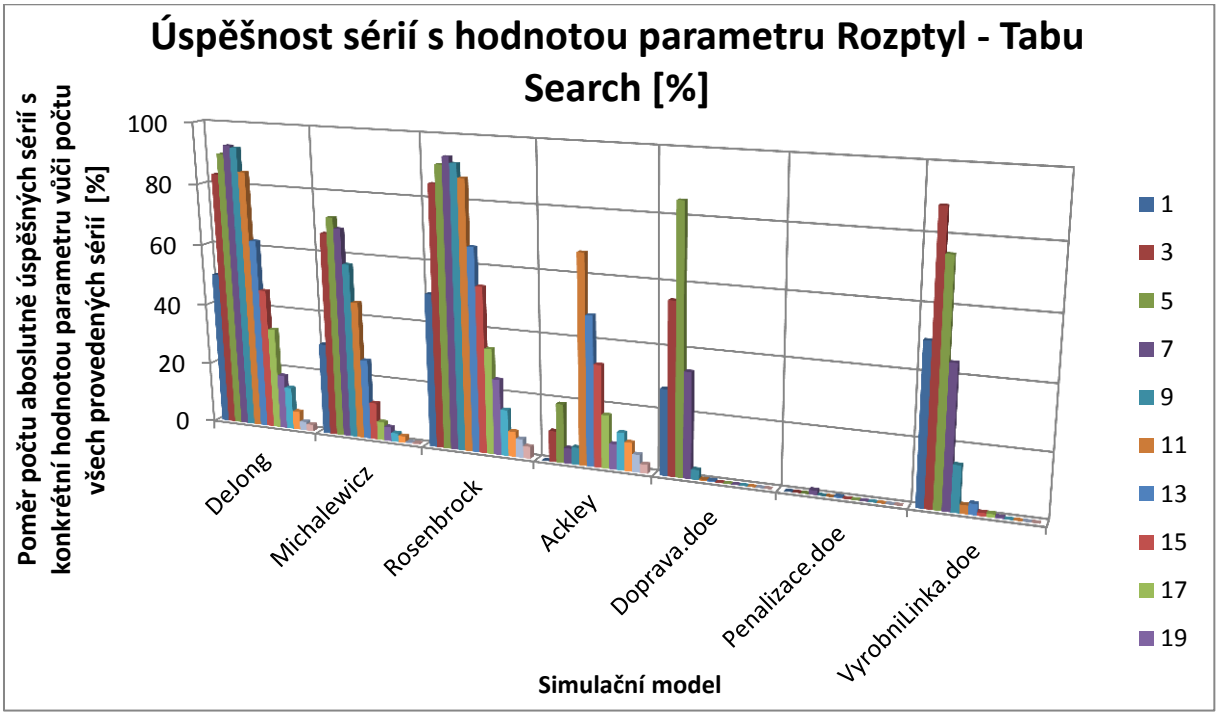
6.3.2 Velikost populace



Obr. 6-8 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Hill Climbing

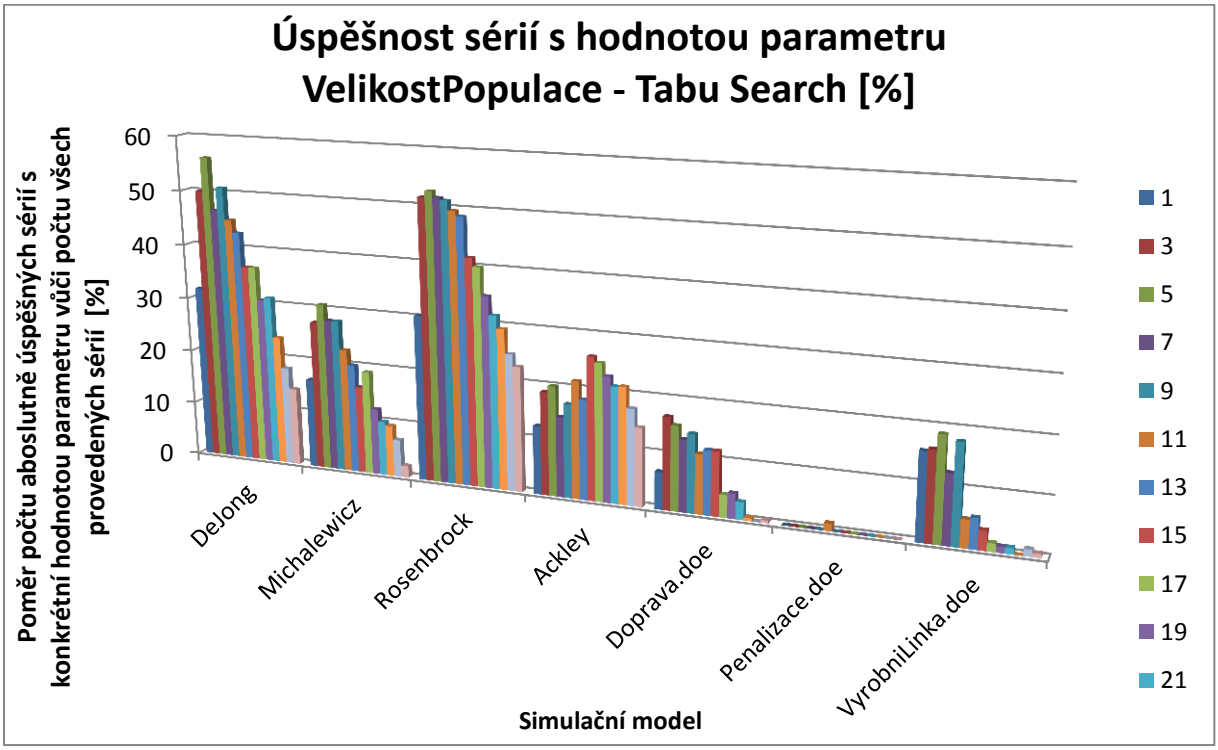
6.4 Tabu Search

6.4.1 Rozptyl



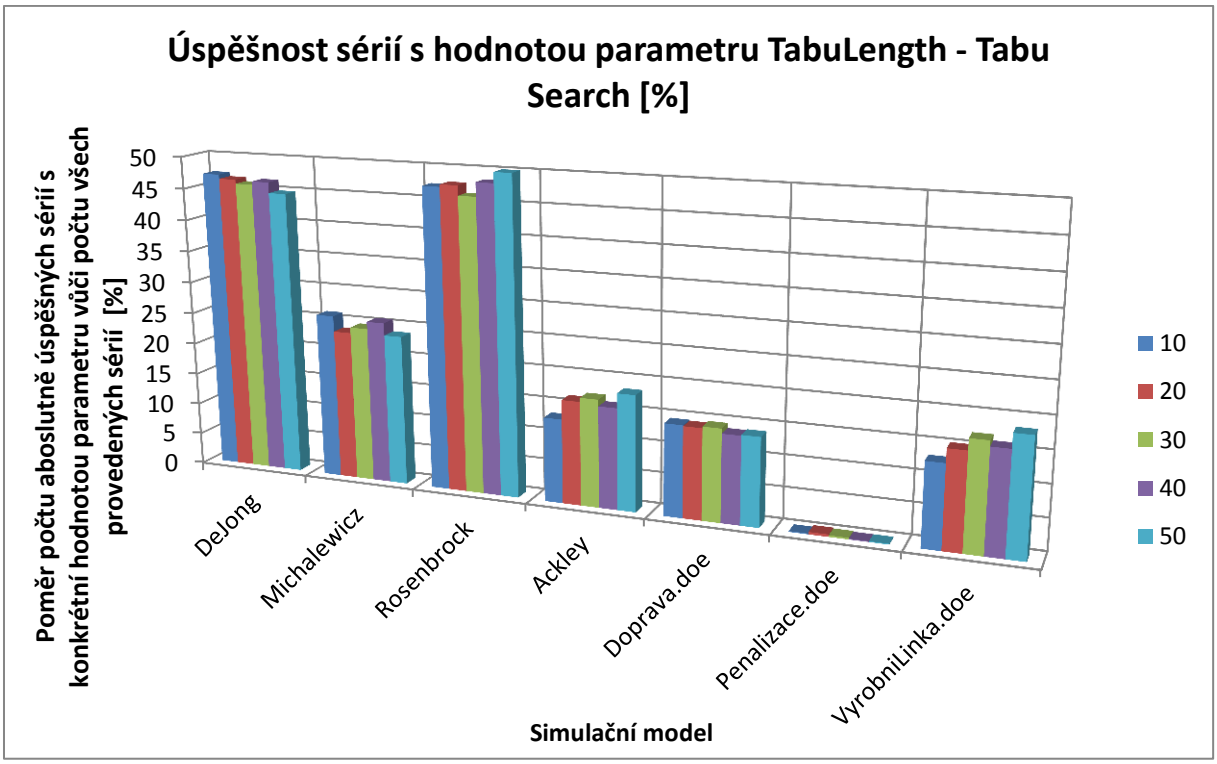
Obr. 6-9 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Tabu Search

6.4.2 Velikost populace



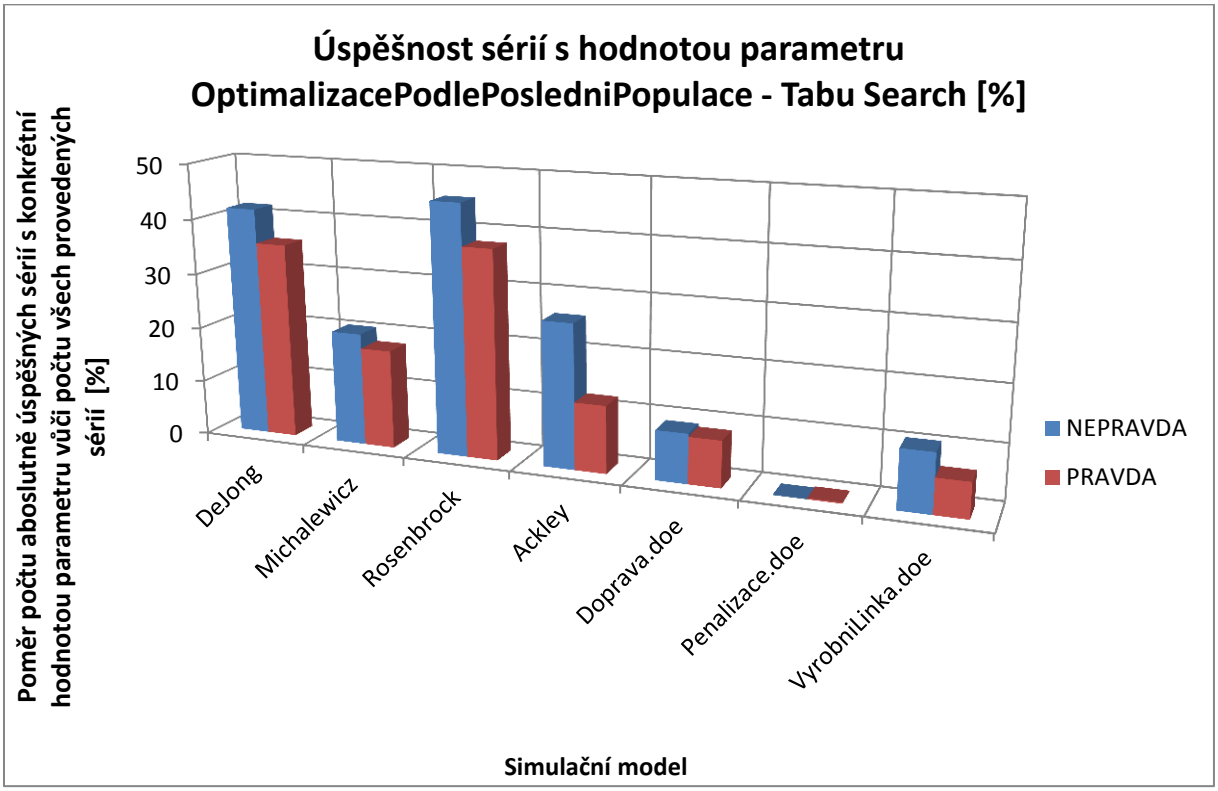
Obr. 6-10 Úspěšnost sérií s hodnotou parametru „Velikost populace“ - Tabu Search

6.4.3 Délka zakázaného seznamu



Obr. 6-11 Úspěšnost sérií s hodnotou parametru „Tabu Length“ - Tabu Search

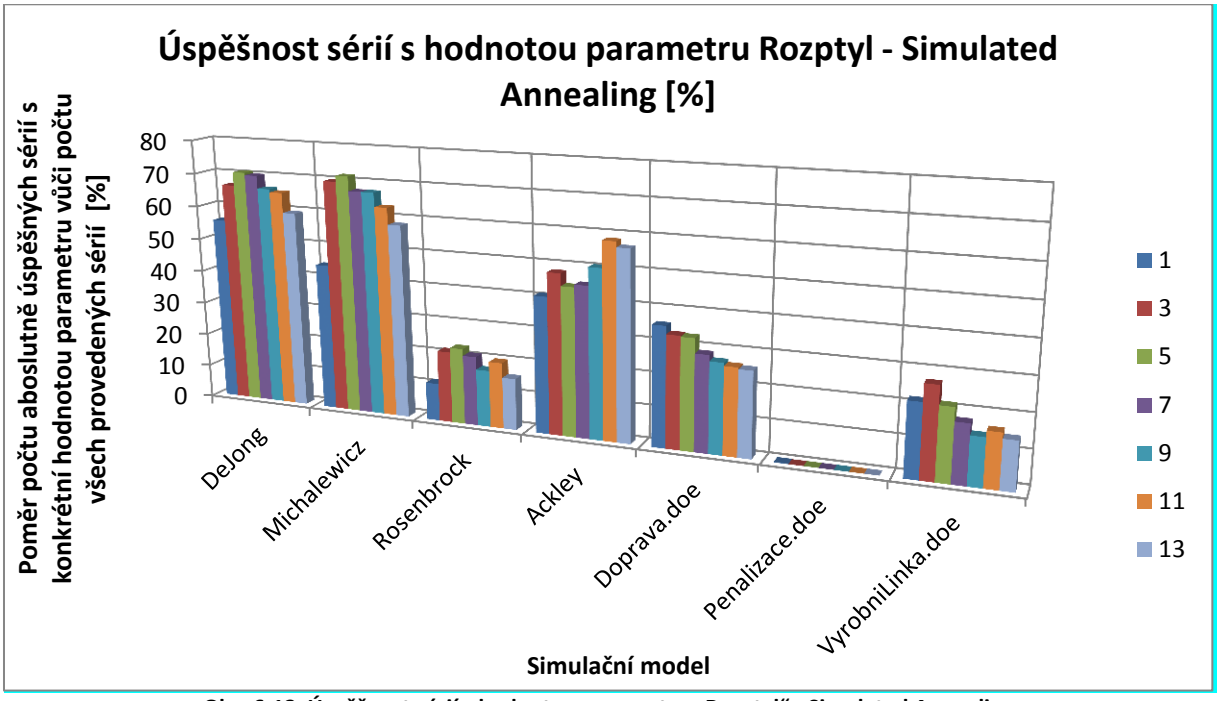
6.4.4 Optimalizace podle poslední populace



Obr. 6-12 Úspěšnost sérií s hodnotou parametru „Optimalizace podle poslední populace“ - Tabu Search

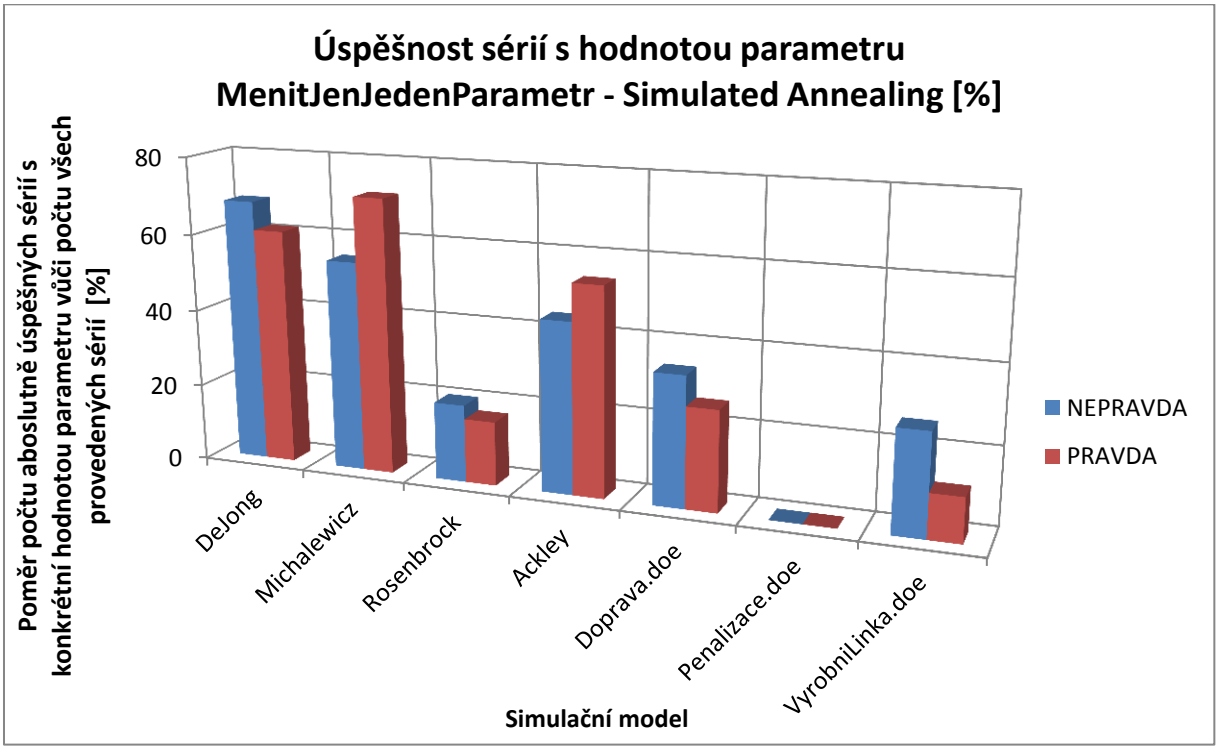
6.5 Simulated Annealing

6.5.1 Rozptyl



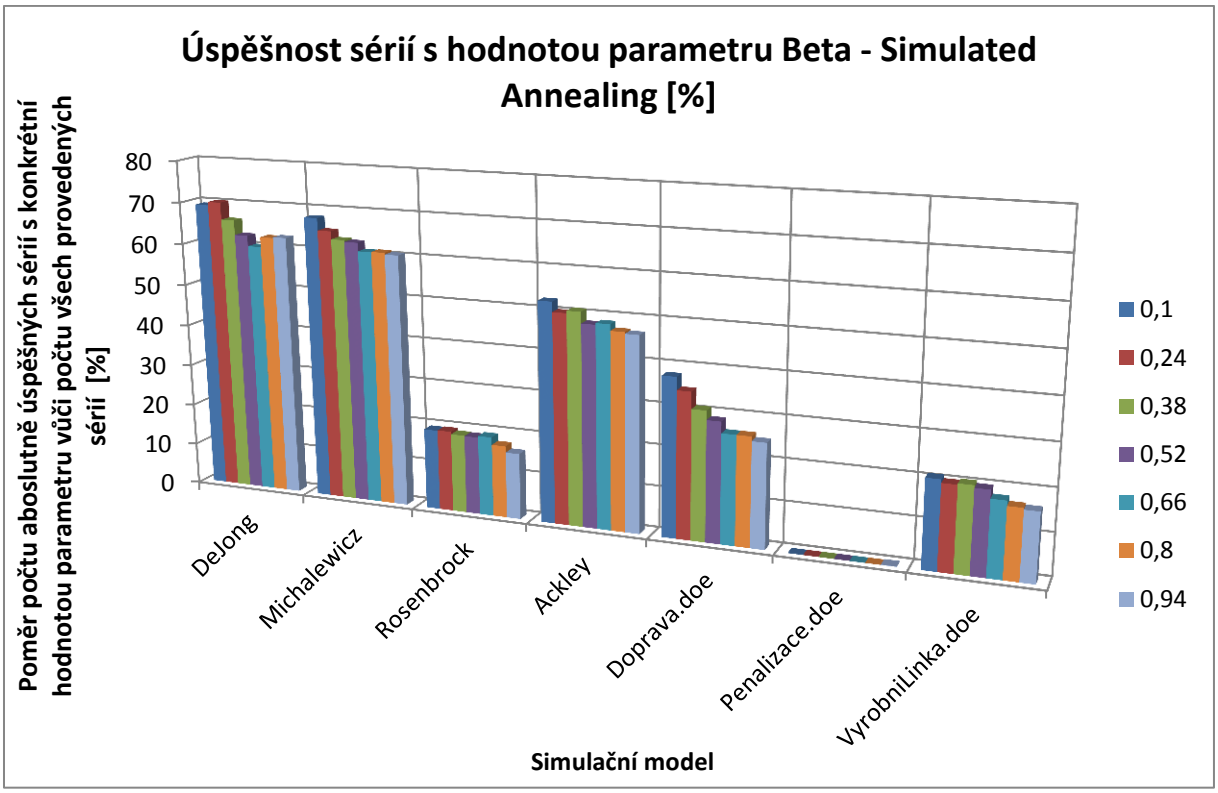
Obr. 6-13 Úspěšnost sérií s hodnotou parametru „Rozptyl“ - Simulated Annealing

6.5.2 Měnit jen jeden parametr



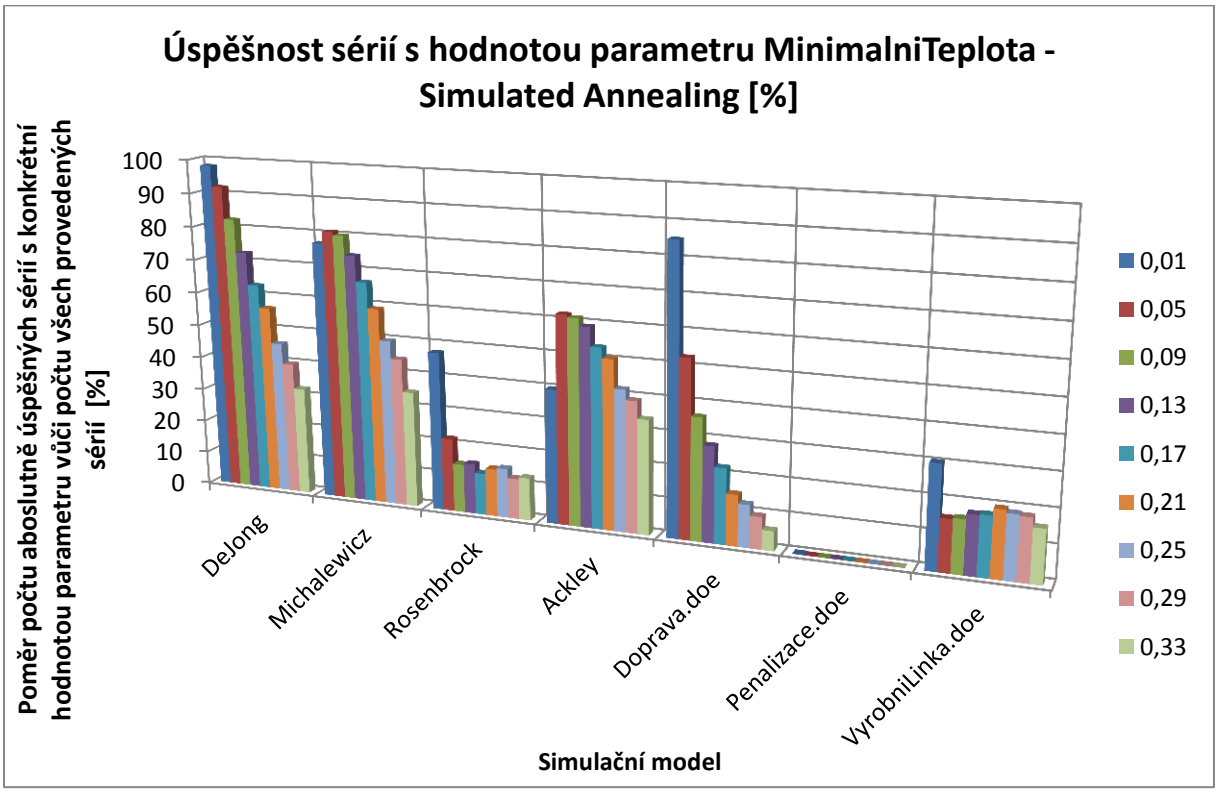
Obr. 6-14 Úspěšnost sérií s hodnotou parametru „Měnit jen jeden parametr“ - Simulated Annealing

6.5.3 Beta



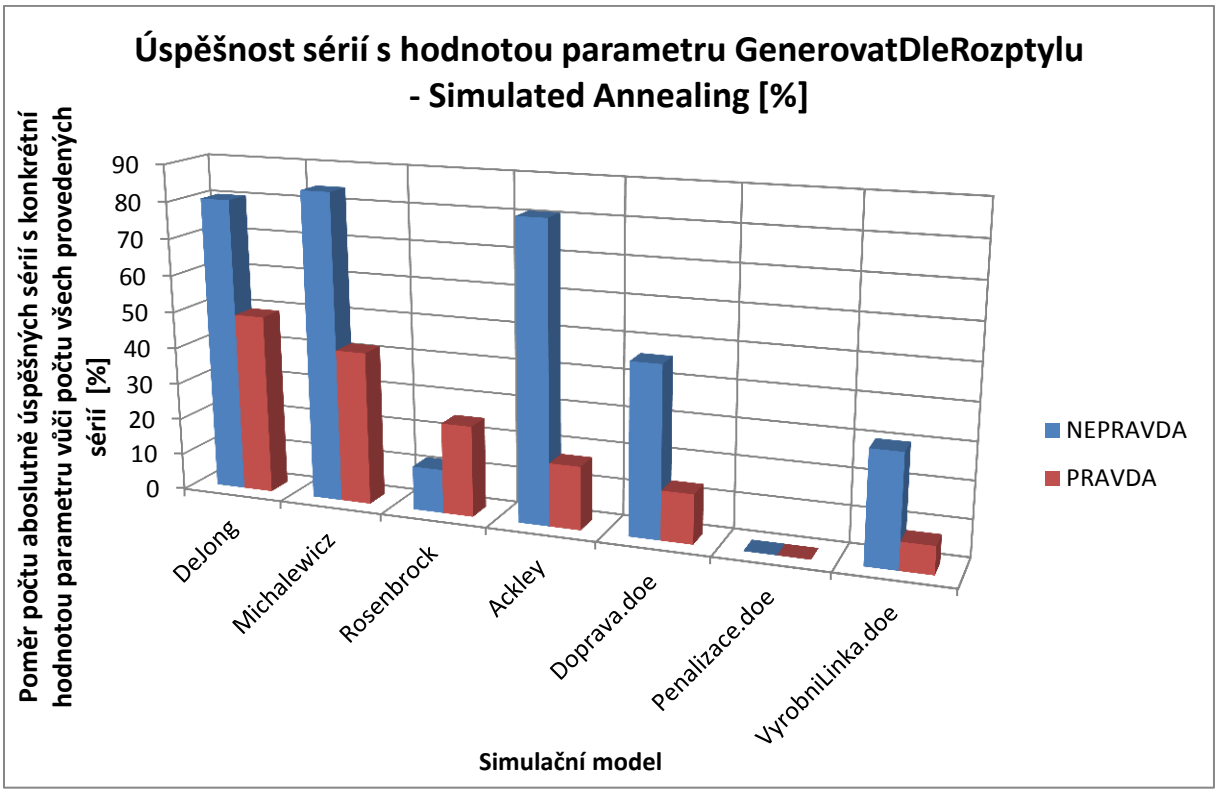
Obr. 6-15 Úspěšnost sérií s hodnotou parametru „Beta“ - Simulated Annealing

6.5.4 Minimální teplota



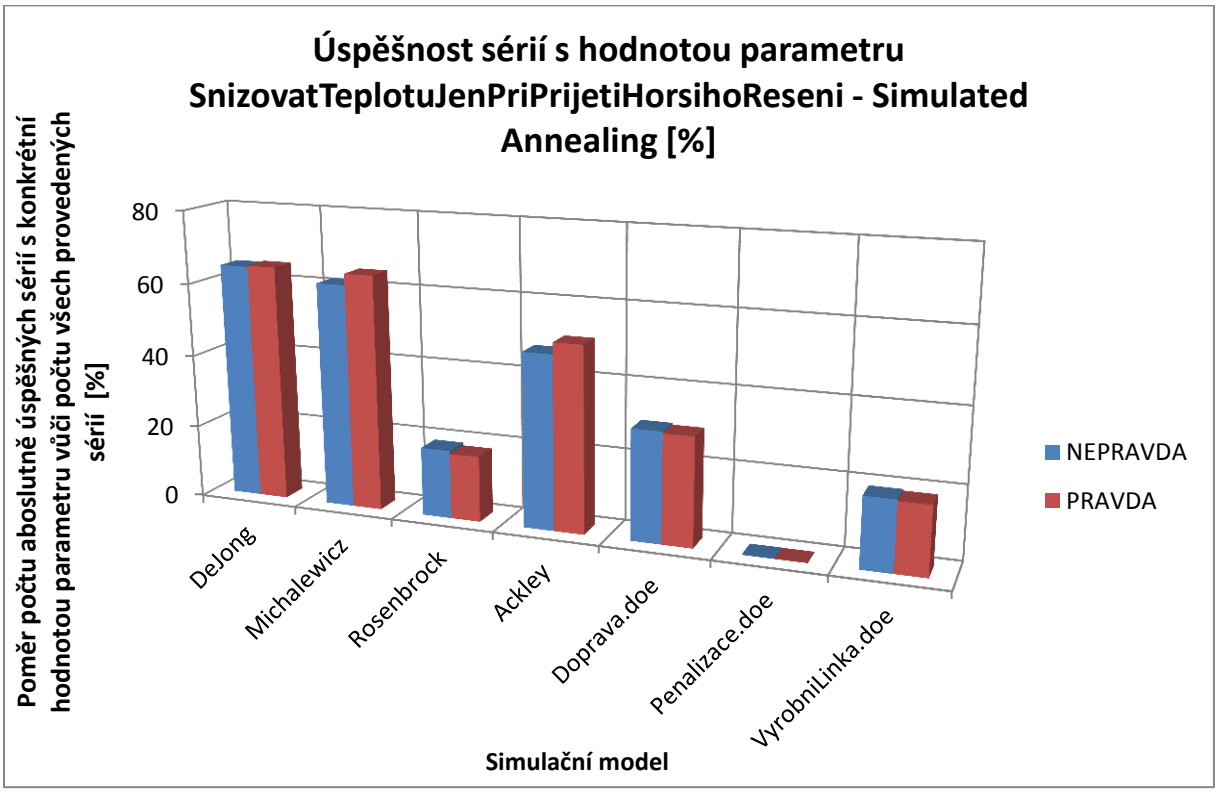
Obr. 6-16 Úspěšnost sérií s hodnotou parametru „Minimální teplota“ - Simulated Annealing

6.5.5 Generovat dle rozptylu



Obr. 6-17 Úspěšnost sérií s hodnotou parametru „Generovat dle rozptylu“ - Simulated Annealing

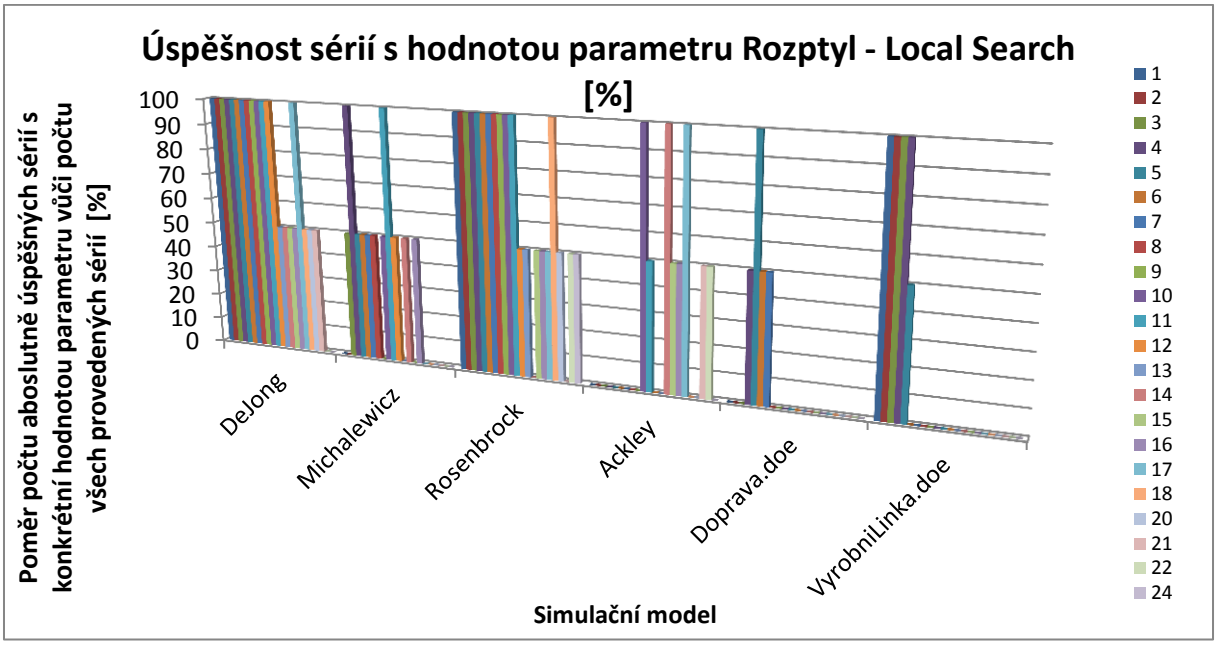
6.5.6 Snižovat teplotu jen při přijetí horšího řešení



Obr. 6-18 Úspěšnost sérií s hodnotou parametru „Snižovat teplotu jen při přijetí horšího řešení“ - Simulated Annealing

6.6 Local Search

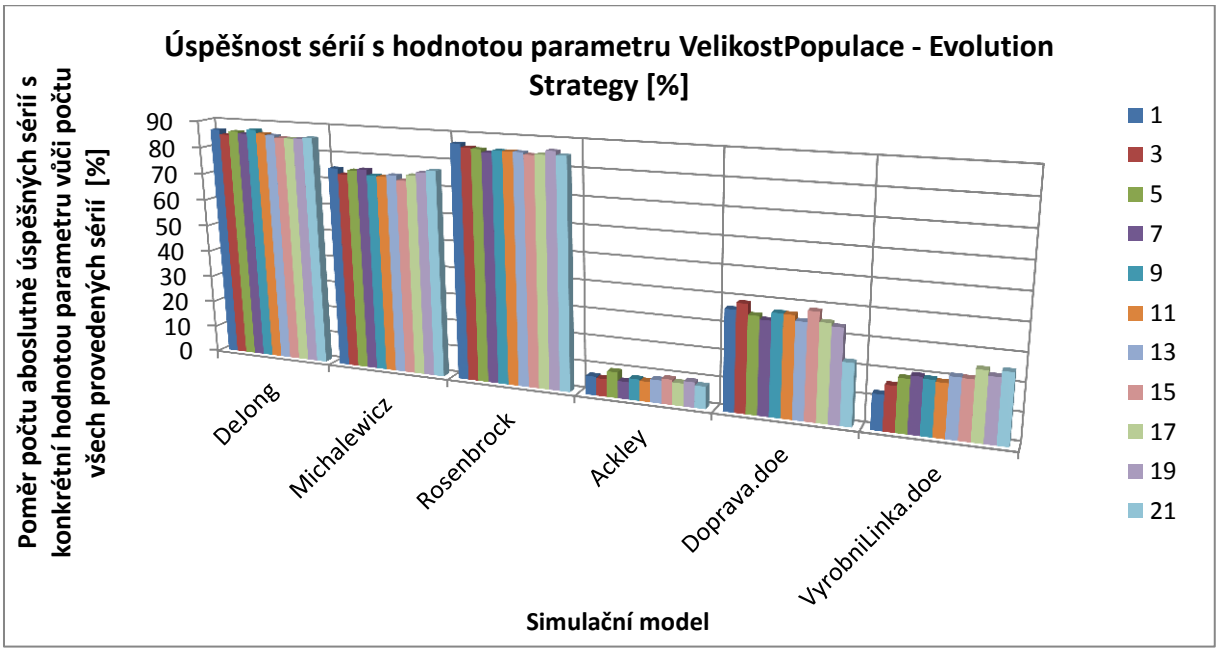
6.6.1 Rozptyl



Obr. 6-19 Úspěšnost sérií s hodnotou parametru „Rozptyl" – Local Search

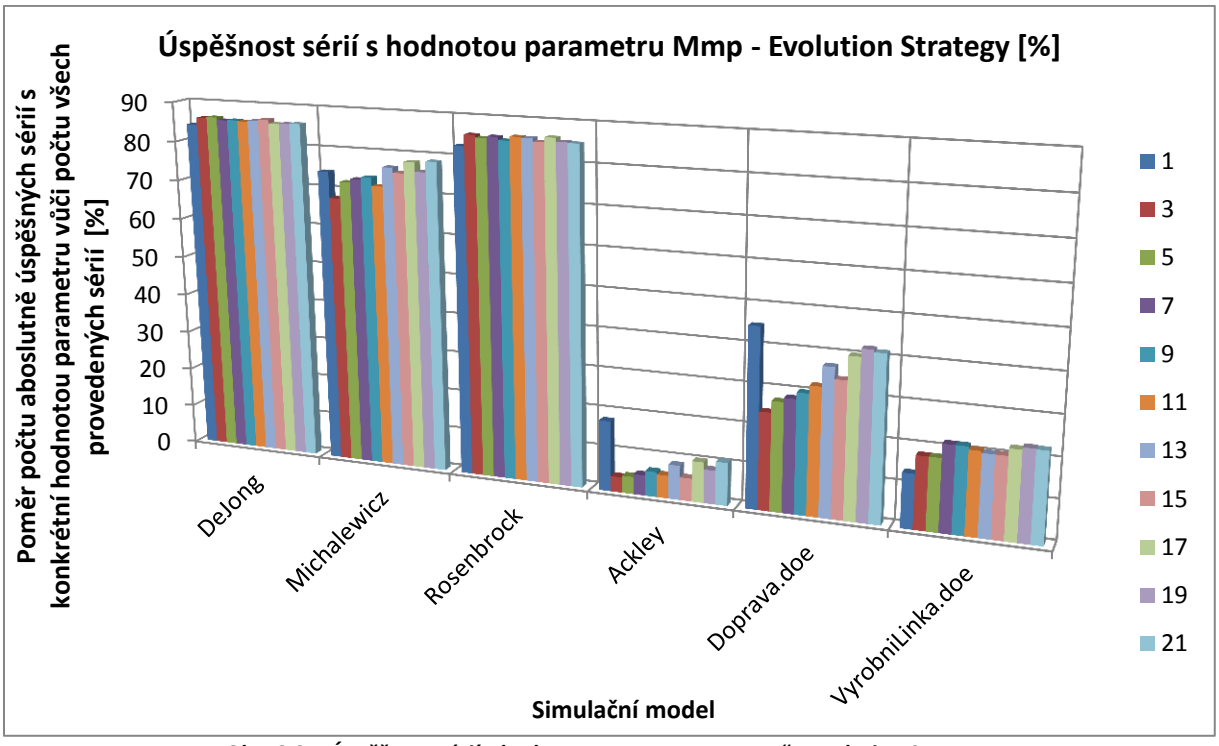
6.7 Evolution Strategy

6.7.1 Velikost populace



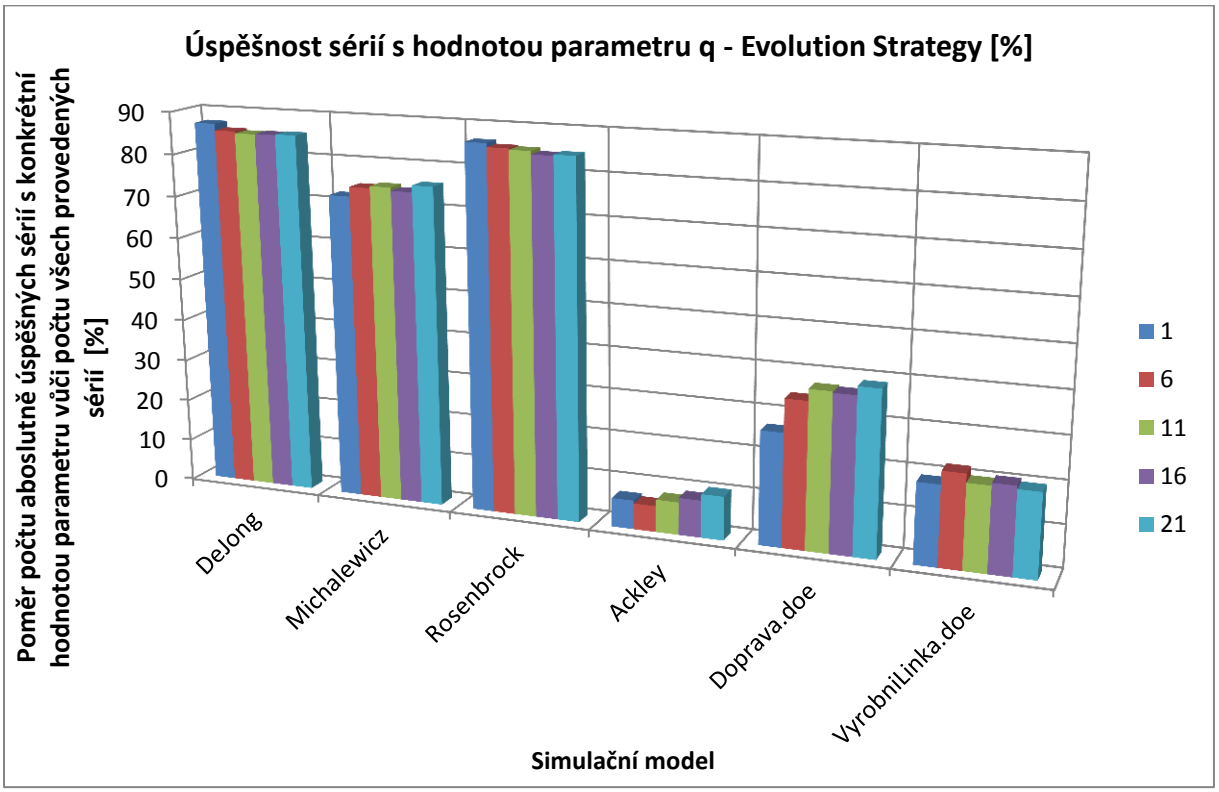
Obr. 6-20 Úspěšnost sérií s hodnotou parametru „Velikost populace" – Evolution Strategy

6.7.2 Mmp – počet potomků



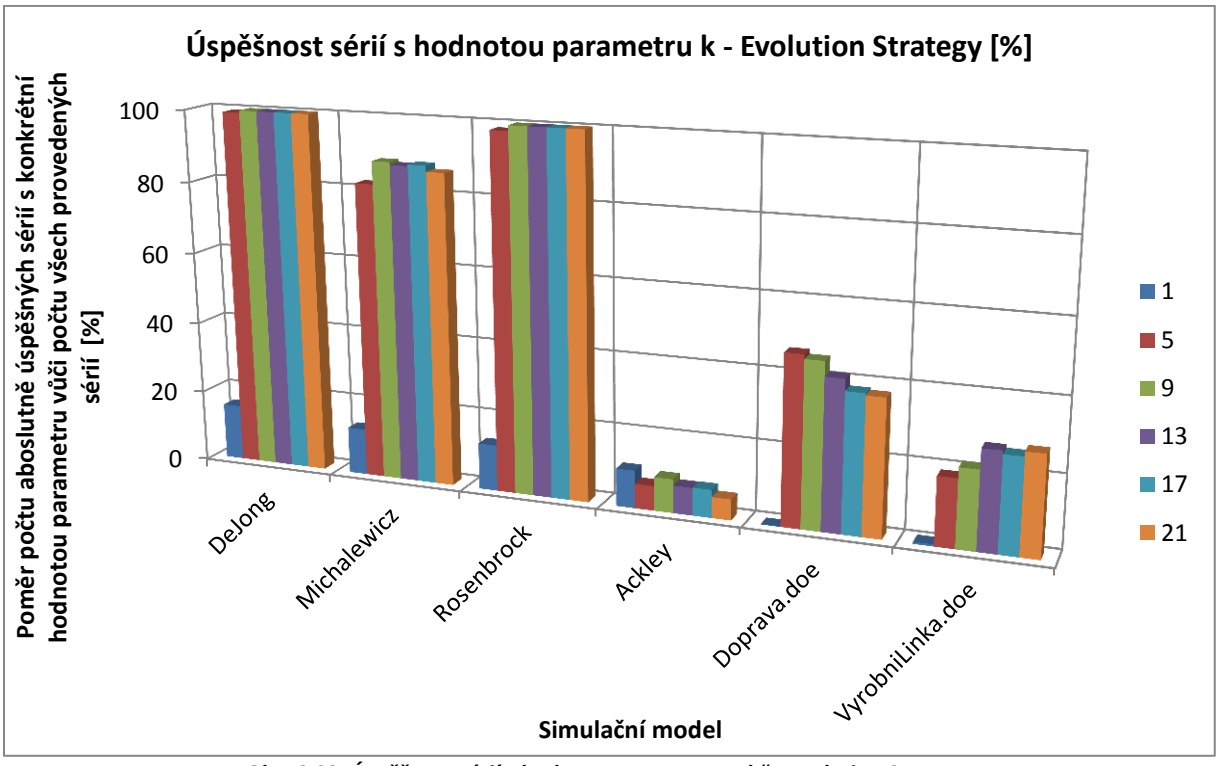
Obr. 6-21 Úspěšnost sérií s hodnotou parametru „Mmp“ - Evolution Strategy

6.7.3 q – počet předcházejících úspěchů



Obr. 6-22 Úspěšnost sérií s hodnotou parametru „q“ - Evolution Strategy

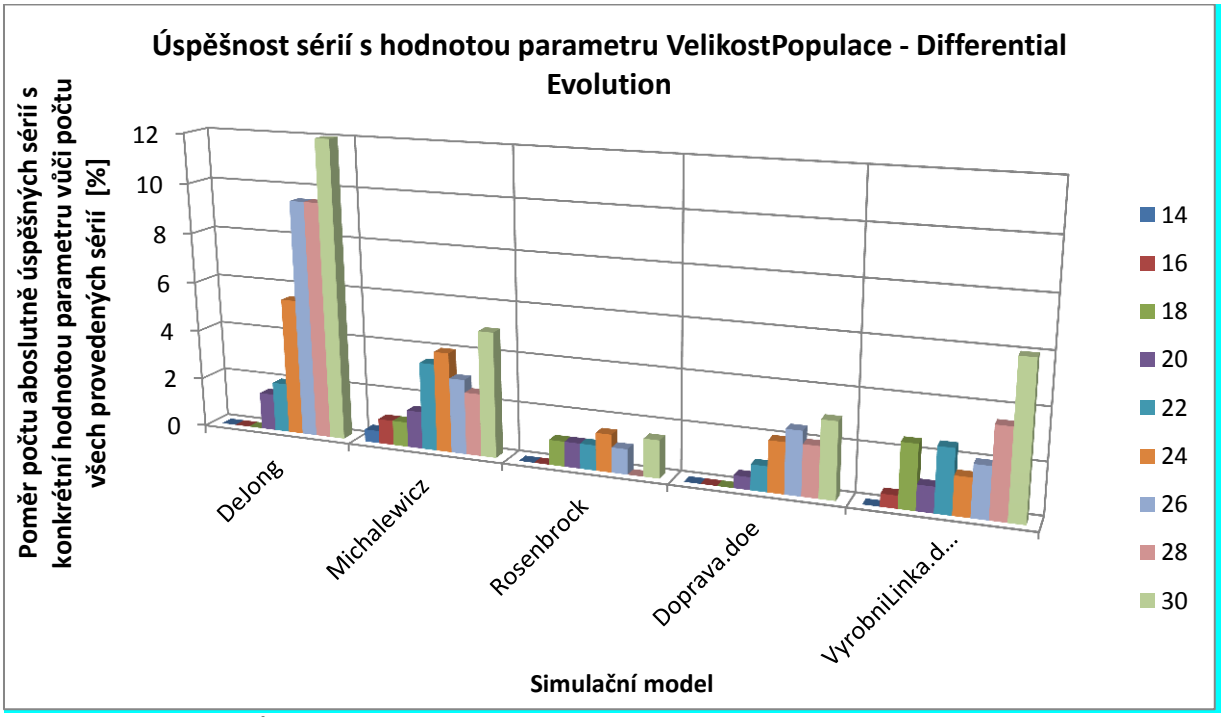
6.7.4 k – počet jedinců v turnaji



Obr. 6-23 Úspěšnost sérií s hodnotou parametru „k“ - Evolution Strategy

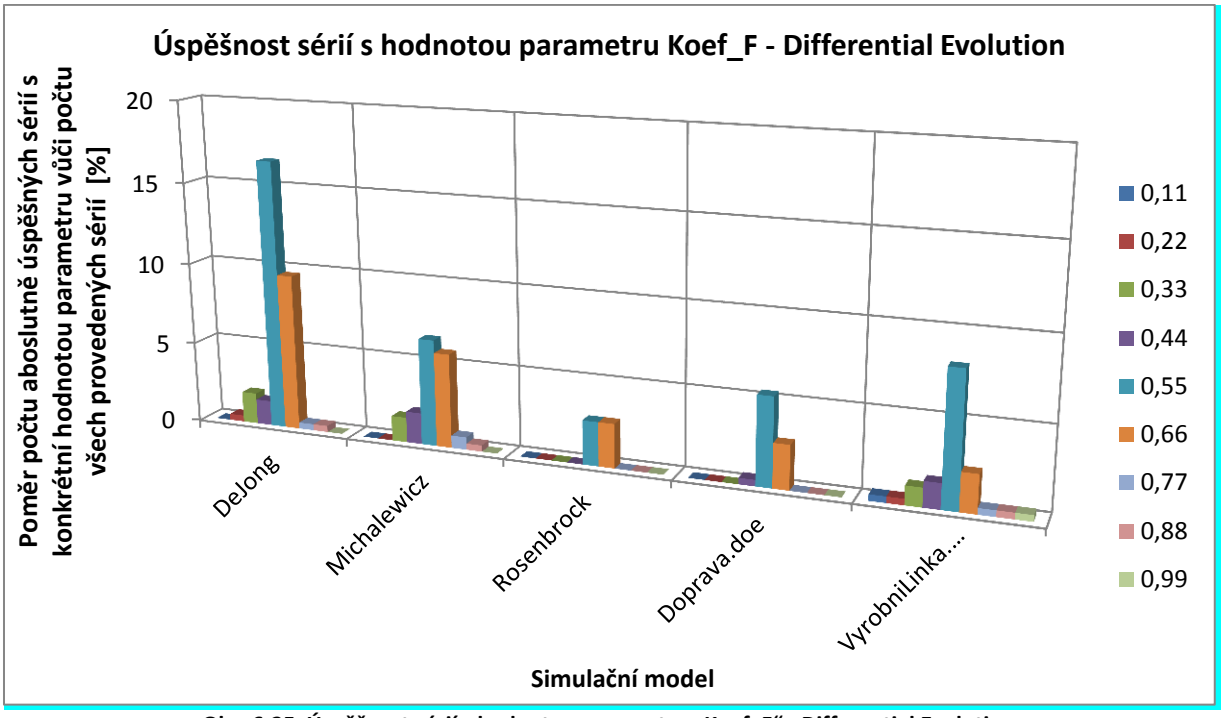
6.8 Differential Evolution

6.8.1 Velikost populace



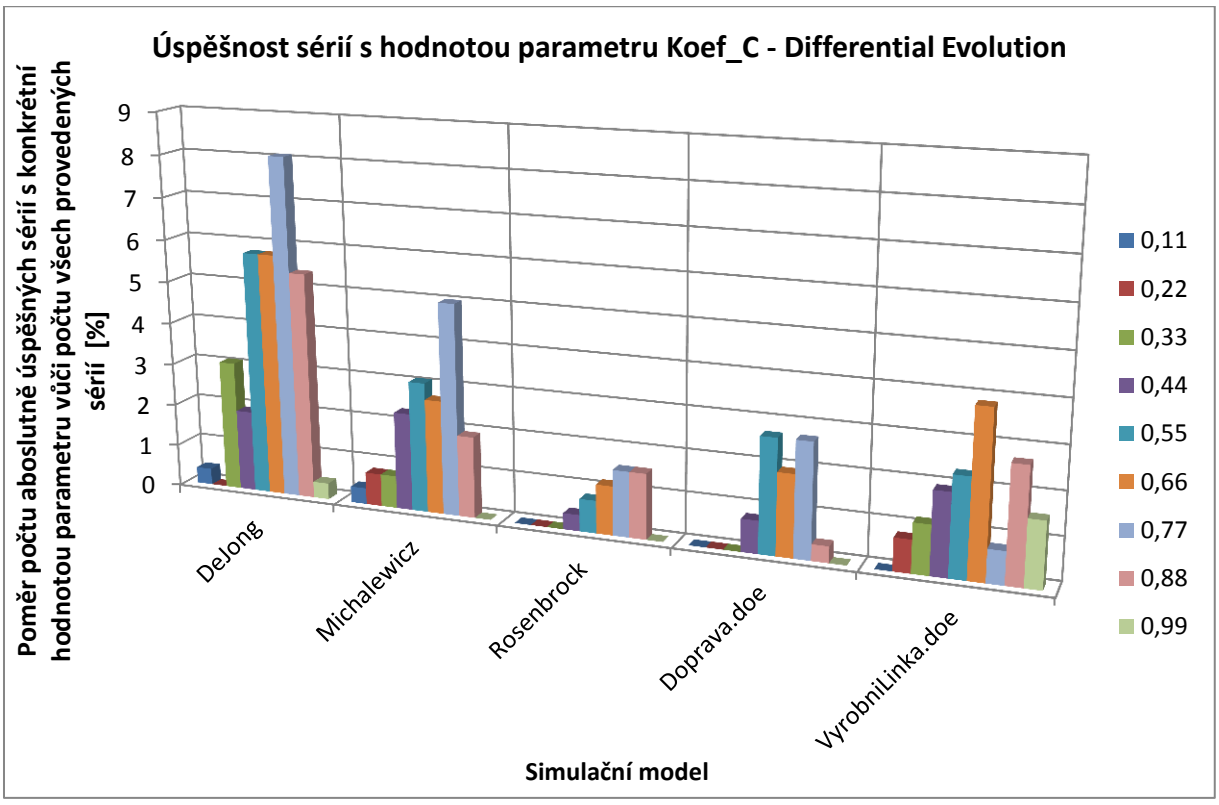
Obr. 6-24 Úspěšnost sérií s hodnotou parametru „VelikostPopulace“ – Differential Evolution

6.8.2 Koef_F – koeficient adaptivního pravidla



Obr. 6-25 Úspěšnost sérií s hodnotou parametru „Koef_F“ - Differential Evolution

6.8.3 Koef_C – pravděpodobnost nahrazení souřadnic rozhodovacích proměnných



Obr. 6-26 Úspěšnost sérií s hodnotou parametru „Koef_C“ - Differential Evolution