University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Generátor trojúhelníkových sítí zadaných vlastností brutální silou

Research Report

Tomáš Hlavatý, Václav Skala

Technical Report No. DCSE/TR-2002-09
March, 2002

Distribution: public

Technical Report No. DCSE/TR-2002-09
March 2002

Generátor trojúhelníkových sítí zadaných vlastností brutální silou

Tomáš Hlavatý, Václav Skala

Abstract

Many heuristic algorithms searching for triangulations by a given criterion exist. The main problem of these algorithms is that the found solution is only an approximation with some error. The size of the error cannot be known without exact solution, which can be found only by brutal force. This paper presents an algorithm, which generates the exact solution for the given criterion by brutal force. Unfortunately the time complexity of the algorithm is generally non-polynomial (NP complexity). In this paper a series of techniques (hash table, preprocessing, using parallel and distribution processing) for decreasing the time of the computation are presented.

This work was supported by the Ministry of Education of the Czech Republic - project MSM23500005

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

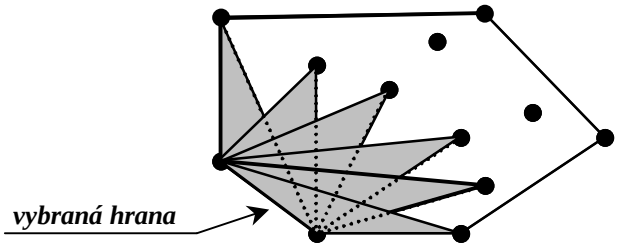
Obsah

1. Úvod

Při řešení problému nalezení trojúhelníkové sítě, kde máme zadány pouze body reprezentující vrcholy trojúhelníků (v E^2) a kritérium, kterému musí daná trojúhelníková síť optimálně vyhovovat, dostaneme se obecně na řešení NP problému. Sice již pro některá kritéria existují algoritmy s polynomiální složitostí, ale stále se jedná o dosti malou skupinu. Jednou z možností, jak v dostupném čase daný problém řešit obecně, je využití heuristik, které sice nezaručují nalezení optima, ale pouze se k němu s určitou chybou blíží. Dá se říci, že právě velikost této chyby by se dala použít jako velmi dobré kritérium při porovnávání jednotlivých heuristických metod. Ovšem je tu dosti závažný problém. Pro stanovení velikosti této chyby potřebujeme znát dané optimum a to bohužel neznáme. Toto je také hlavním důvodem, proč se daným problémem zabývat.
Hlavním cílem této práce je nalézt obecný algoritmus, který bude schopen vytvořit trojúhelníkovou síť, která by optimálně vyhovovala zvolenému kritériu. Je především kladen důraz na rychlost algoritmu (s ohledem, že se jedná o NP problém) a jeho obecnost, aby se dal použít pro libovolné kritérium.

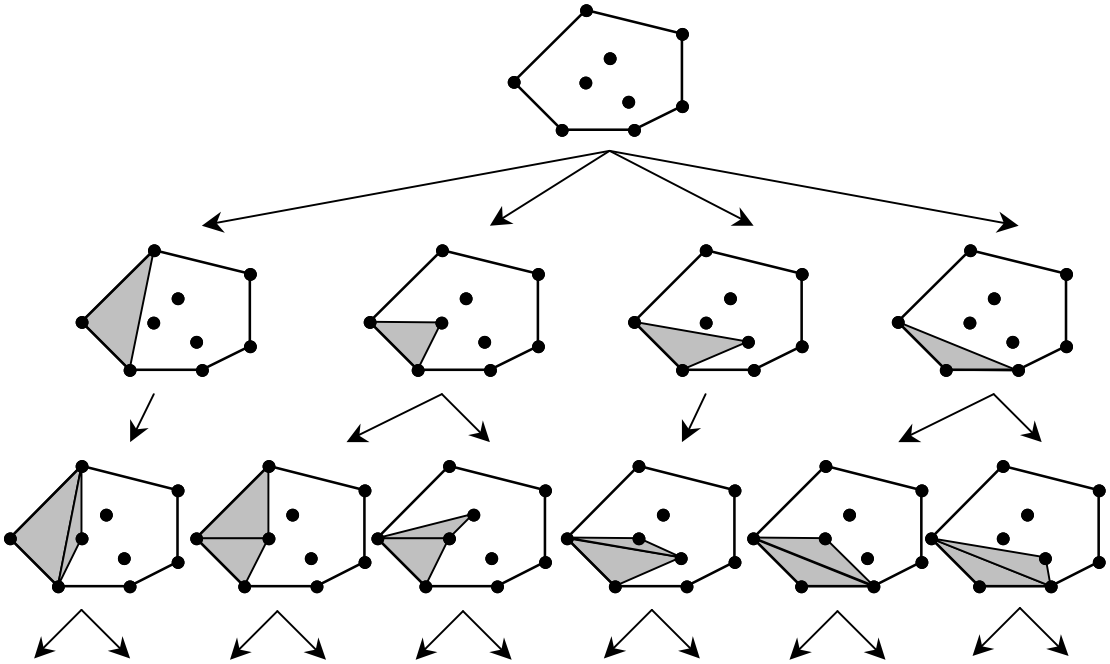
2. Metoda vkládání trojúhelníků

V diplomové práci (viz [Hla01]) byla uvedena řada metod, které hledají trojúhelníkové sítě podle již zmíněných podmínek v úvodu. Při jejich porovnání se jevila jako nejlepší tzv. *Metoda vkládání trojúhelníků*, z které také vychází tato práce. Pro úplnost je v této kapitole tato metoda v krátkosti přiblížena.
Prvním krokem při návrhu této metody byl náhled na trojúhelníkovou síť (dále jen *triangulaci*) jako na množinu trojúhelníků. Další úvaha byla především zaměřena na otázku, jak vygenerovat všechny možné triangulace, které lze nad danou množinou bodů sestrojit, aniž by vznikly duplicity a aniž by byla nějaká opomenuta. Je třeba brát v úvahu, že je hledáno globální optimum pro zvolené kritérium. Pro jeho získání je třeba projít všechny možné kombinace a z nich vybrat tu nejlepší. Vznikem duplicit by se ztratila rychlost výpočtu, což není v tomto případě přípustné, a opomenutím některých triangulací by hrozilo, že by mohlo být nalezeno chybné řešení.
Počátečním krokem tohoto algoritmu je nalezení konvexní obálky, která bude představovat v našem případě polygon, který ohraničuje část, do které je třeba vložit trojúhelníky, aby vznikla tzv. *úplná triangulace* (tj. síť skládající se pouze z trojúhelníků, které se vzájemně neprotínají). Z tohoto polygonu může být nyní vybrána libovolná hrana a přidán tzv. *prázdný trojúhelník* (trojúhelník, který neobsahuje žádné vnitřní body ze vstupní množiny bodů), který bude obsahovat právě námi zvolenou hranu. Ovšem těchto prázdných trojúhelníků může být hnedka několik (viz obrázek 2.1).



Obr. 2.1 Příklad všech prázdných trojúhelníků obsahujících vybranou hranu

Jelikož je potřeba vygenerovat všechny triangulace, musí být zajištěno, aby výběrem trojúhelníků nedošlo k opomenutí některých kombinací. K tomuto lze použít datovou strukturu představují tzv. *strom,* kterou si lze prohlédnout na obrázku 2.2.



Obr. 2.2 Příklad vytváření stromu postupným vkládáním trojúhelníků

Jak je z tohoto obrázku vidět, z každého uzlu stromu vychází tolik větví, kolik lze vložit prázdných trojúhelníků, které splňují následující podmínky:

- Obsahují předem vybranou hranu, která je zároveň součástí polygonu ohraňující neztriangulovanou oblast.

- Celá plocha reprezentující obsah trojúhelníků je vnořena v oblasti, která ještě nebyla ztriangulována.

Pokud je toto splněno, pak listy vytvořeného stromu reprezentují jednotlivé triangulace a tedy i náš cíl. Po nalezení těchto triangulací, není již problém je ohodnotit a zapamatovat si tu triangulaci, která bude mít nejlepší ohodnocení.
Algoritmus, který by dokázal vygenerovat resp. prohledat strom podle uvedených podmínek je bodově popsán na obrázku 2.3. Pro procházení (a zároveň i generování) stromu je použita slepá strategie tzv. *procházení stromu do hloubky* - stromem se prochází postupně z jedné strany na druhou a je použita datová struktura *zásobník*, ve které je uložena pouze aktuální cesta říkající, kde se momentálně daný algoritmus ve stromě nachází.

1. Ke všem hranám vytvořme seznamy všech prázdných trojúhelníků, které příslušnou hranu obsahují
2. Nalezněme hrany konvexní obálky
3. Vytvořme zásobník pro uložení informací potřebných k znovuobnovení již vytvořeného stavu
4. Vytvořme pole **T**, do kterého se bude zapisovat výsledná triangulace a pole **setP**, ve kterém budou uloženy hraniční polygony ještě znetriangul. oblasti
5. Do pole **setP**, vložme konvexní obálku a pro vybranou hranu nastavme ukazatel na první prázdný trojúhelník obsahující příslušnou hranu
6. Uložme **setP** do zásobníku
7. Vyzkoušejme zda trojúhelník, na který ukazuje ukazatel, lze vložit do triangulace (tj. neprotíná se s hranami polygonu v **setP**), pokud ne, zkusme další nalezené trojúhelníky ze seznamu pro příslušnou hranu
8. **IF** podařilo se najít trojúhelník, který lze vložit **THEN**
    a. Ulož do zásobníku odkaz na další trojúhelník v seznamu
    b. Vlož do pole **T** hrany trojúhelníku, které v něm ještě nejsou
    c. Ulož do zásobníku počet vložených hran do **T**
    d. Zaznamenej změny v množině polygonů **setP**, které nastali vložením trojúhelníku do vytvářené triangulace

(pokračování na další stránce ...)

e. **IF** triangulace je kompletní **THEN**

- Pošli **T** na výstup
- Odeber poslední vkládané hrany do **T**
- Načti ze zásobníku **setP** a ukazatel na trojúhelník, který se má vložit do triangulace

ELSE

- Vyber ze **setP** polygon a hranu, s kterou se bude pracovat
- Nastav ukazatel na první tzv. prázdný trojúhelník, který obsahuje vybranou hranu ze **setP**

ELSE

a. Ulož do zásobníku odkaz na další trojúhelník v seznamu

b. Odstraň poslední uloženou položku ze zásobníku

c. Načti ze zásobníku **setP**, počet naposledy vložených hran do **T** a ukazatel na trojúhelník, který se má vložit do triangulace

d. Odeber z **T** naposledy vložené hrany

9. Skok na bod 6

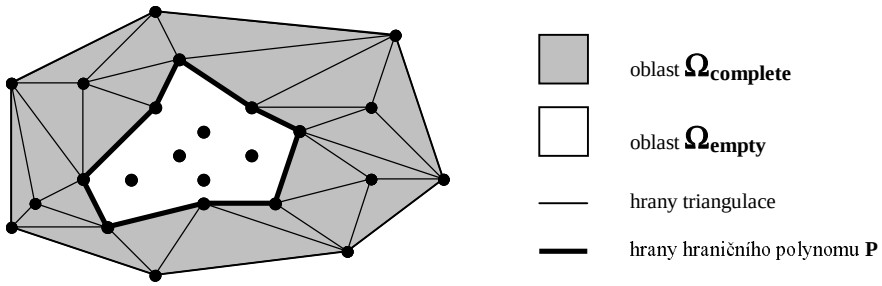
10. Konec algoritmu

Obr. 2.3 Algoritmus metody vkládání trojúhelníků

Tento algoritmus byl implementován pro nalezení MWT (Minimum Weight Triangulation) a s výsledky se lze setkat opět v diplomové práci (viz [Hla01]). Pro představu o rychlosti výpočtu lze konstatovat, že tento algoritmus je schopen na počítači DELL s procesorem 2x650MHz a pamětí 1GB najít řešení pro množiny kolem 20-25 bodů v reálném čase (řádově hodiny).

3. Modifikace metody vkládání trojúhelníků

Metoda vkládání trojúhelníků je schopna pro obecné kritérium najít v dostupném čase řešení pro množinu řádově do 25 bodů. Bohužel pro praktické využití je toto dosti malá množina. Vznikla otázka, co vše lze ještě využít pro urychlení této metody a dosáhnutí řešení pro větší množiny bodů.
Podívejme se na způsob generování triangulací trochu odlišným způsobem. Předpokládejme, že máme množinu bodů a metodou vkládání trojúhelníků je generován výše uvedený strom (viz druhá kapitola, obr. 2.2). Bude-li se nacházet v uzlu stromu, který není ani kořen ani list stromu, reprezentuje takovýto uzel jakousi částečně vytvořenou triangulaci, která může např. vypadat tak, jak je znázorněno na obrázku 3.1.



Obr. 3.1 Rozdělení tzv. neúplné triangulace na oblasti

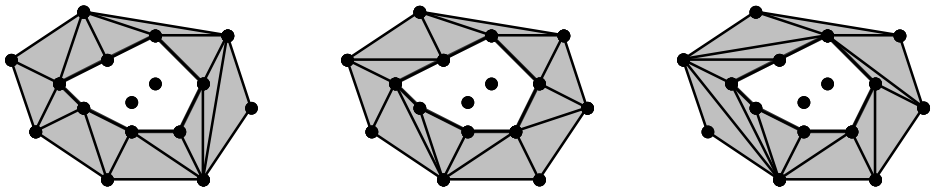
Ta může být rozdělena tzv. *hraničním polygonem* ***P*** (viz obrázek) na dvě oblasti:

- $\Omega_{complete}$ … oblast, která již byla triangulována (tj. tuto oblast tvoří již vložené trojúhelníky resp. hrany)

- Ω_{empty} …oblast, do které je třeba ještě vložit trojúhelníky, aby vznikla tzv. úplná triangulace

Je třeba ještě poznamenat, že tyto oblasti obecně nemusí být spojité (mohou být rozděleny do více nespojitých oblastí), ale pro jednoduchost úvahy bude toto zatím opominuto.
Stejným pohledem se lze dívat i na kořen a listy stromu, které byly předtím opomenuty. Kořen generovaného stromu představuje neúplnou triangulaci obsahující pouze hrany konvexní obálky. Hraničním polygonem ***P*** je zde konvexní obálka, tj. oblast Ω_{empty} tvoří plocha ohraničená konvexní obálkou a oblast $\Omega_{complete}$ tvoří tzv. prázdný prostor (ještě nebyl vložen žádný trojúhelník pro zkonstruování triangulace).
Naopak list stromu představuje tzv. *úplnou triangulaci*. Hraniční polygon ***P*** je již v tomto uzlu reprezentován prázdnou množinou (není již co triangulovat), tj. Ω_{empty} oblast je také prázdná a oblast $\Omega_{complete}$ představuje sjednocení ploch všech vložených trojúhelníku (vnitřek konvexní obálky).
Celkově lze říci, že výše zavedené definice oblastí mohou být použity pro libovolný uzel stromu. Při tomto pohledu na generovaný strom, lze zjistit, že v některých

případech jsou jednotlivé oblasti shodné, i když se jedná o různé uzly stromu. Pro názornost je na obrázku 3.2 znázorněn příklad, jak takováto situace může vypadat.



a) ***b)*** ***c)***

Obr. 3.2 Příklad neúplných triangulací, které mají stejný hraniční polygon ***P***

Bude-li se na tyto uzle dívat s pohledu metody vkládání trojúhelníků, lze konstatovat, že stromy (nebo spíše podstromy), které vzniknou vygenerováním z daných uzlů všech možných kombinací trojúhelníků pro vytvoření úplné triangulace, jsou totožné. Při hledání optima, pak lze na ohodnocující funkci *w(T)*, kde *T* je množina hran reprezentující triangulaci zadané množiny bodů, nahlížet podle následující rovnice:

$$w(T) = w(T_{complete}) + w(T_{empty}) \qquad (3.1)$$

kde T_{empty} je množina hran, které byly dohledány z počátečního stavu (hrany počáteční ještě neztriangulované oblasti Ω_{empty}) a $T_{complete}$ je množina zbylých hran tvořící doplněk do množiny *T*. Jednodušeji by se dalo říci, že pro daný výchozí uzel reprezentující neúplnou triangulaci, hrany $T_{complete}$ jsou hrany, které jsou v oblasti $\Omega_{complete}$ (včetně hran hraničního polygonu ***P***), a hrany T_{empty} jsou zbylé hrany, které byly dohledání při triangulaci oblasti Ω_{empty}. Pro tyto množiny hran platí:

$$T_{empty} \cup T_{complete} = T;\ T_{empty} \cap T_{complete} = \emptyset \qquad (3.2)$$

Jak již bylo řečeno, cílem celé úlohy je najít triangulaci, která by optimalizovala zvolené kritérium, tj. je hledána triangulace představující globální optimum (může se jednat jak o minimum tak i o maximum) ohodnocující funkce *w*. Bude-li výchozím bodem libovolný uzel, který tvoří částečnou triangulaci (např. viz obr. 3.2), řešení úlohy pro daný uzel by se dalo matematicky zapsat následujícím způsobem (viz rovnice 3.1):

$$w(T) = w(T_{complete}) + opt\{w(T^{i}_{empty})\}_{\forall i} \qquad (3.3)$$

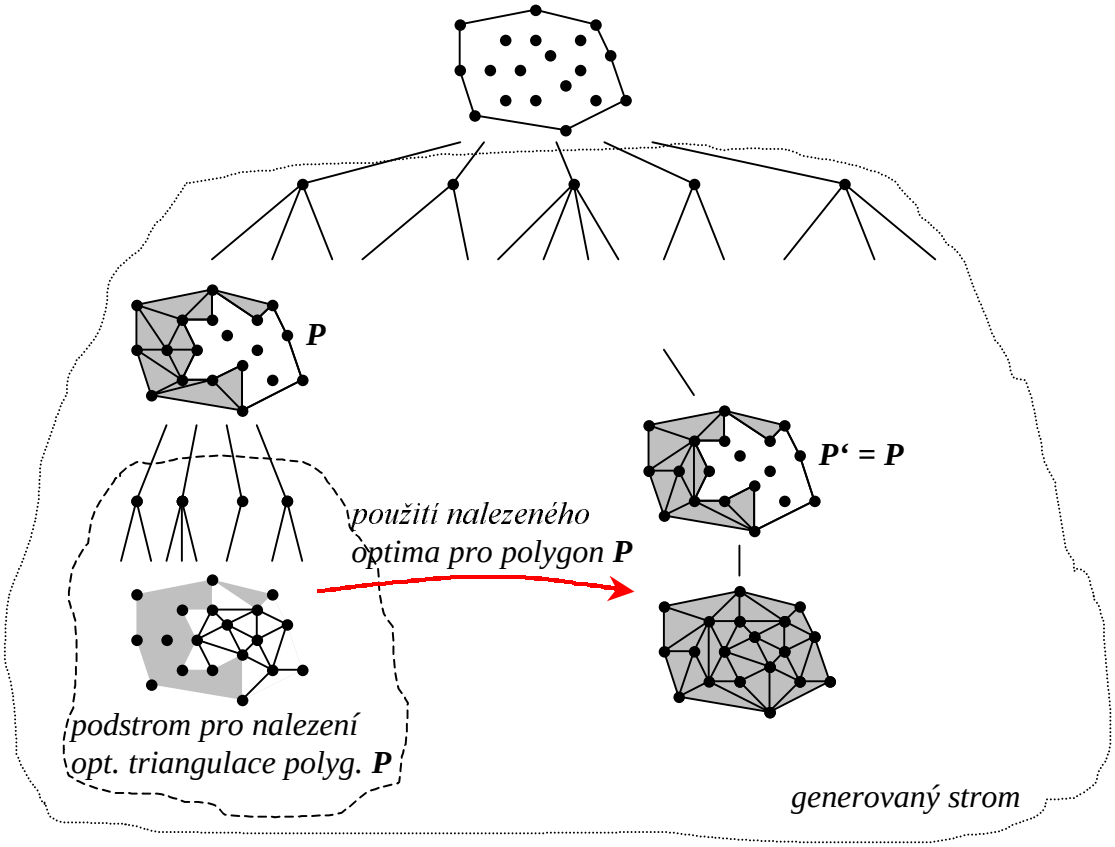
kde množiny T^{i}_{empty} představují všechny možné kombinace hran, které mohou nastat při triangulaci dané množiny Ω_{empty} (opět použita metoda vkládání trojúhelníků) a funkce *opt* vybírá tu nejlepší kombinaci hran, která optimalizuje zvolené kritérium v dané oblasti Ω_{empty} (index *i* je zde použit pouze pro zvýraznění, že kombinací hran pro vytvoření triangulace může být více).

Nyní předpokládejme, že budeme generovat další uzly stromu a narazíme na uzel, který bude představovat triangulaci se stejným hraničním polygonem a tedy i oblastmi $\Omega_{complete}$ a Ω_{empty} (viz obr. 3.2). Při triangulaci oblasti Ω_{empty} a výběru kombinace hran

optimalizující zvolené kritérium, zjistíme, že dostáváme stejnou kombinaci hran jako v předchozím případě (viz rovnice 3.3). Na ohodnocení celé triangulace *w(T')* bude mít při porovnání těchto dvou uzlů pouze vliv ohodnocení hran $T_{complete}$. Hrany T_{empty} zde představují jakousi konstantu.

Začíná-li se objevovat pojem konstanta, vícenásobný výskyt v algoritmu, lze si domyslet, že už by se dalo přemýšlet o urychlení. Shrne-li se vše, co zde bylo dosud napsáno, může být způsob, jak urychlit metodu vkládání trojúhelníků, popsán následovně:

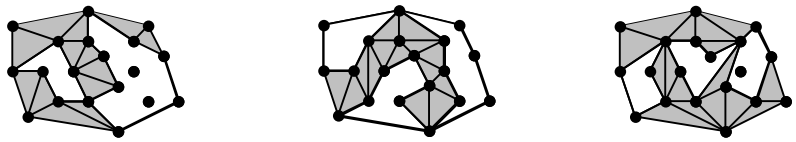
Pro každý uzel generovaného stromu je dán tzv. hraniční polygon *P*, který rozděluje triangulaci do dvou oblastí – oblast, která již tvoří část triangulace, a oblast, do které musí být vloženy hrany (resp. trojúhelníky), aby vznikla úplná triangulace vstupní množiny bodů. Cílem úlohy je najít triangulaci, která by optimálně vyhovovala zvolenému kritériu. Vygenerováním resp. projitím celého podstromu od daného uzlu by neměl být žádný problém určit, které hrany tvoří optimum triangulace daného hraničního polygonu *P*. Pokud se tedy tato množina hran uchová zároveň s hraničním polygonem, při dalším výskytu uzlu se stejným hraničním polygonem, může být již této informace využito, aniž by byla potřeba generovat další podstrom. Je zřejmé, že lepší řešení triangulace polygonu P něž bylo původní nelze najít. Názorněji je na následujícím obrázku uveden případ, jak se generovaný strom pozmění.



Obr. 3.3 Znázornění využití již nalezené optimální triangulace hraničního polygonu P

Při jiném náhledu na obr. 3.3 lze říci, že jsme metodu vkládání trojúhelníků rozšířili tím, že už nevkládáme jenom trojúhelníky, ale také i hrany, které tvoří triangulaci

nějakého polygonu. Ovšem je třeba říci, že tyto hrany tvořící triangulaci polygonu museli být již předtím nalezeny. Mohou být tedy vloženy až teprve při druhém, resp. dalším výskytu daného polygonu při procházení stromu.
Ovšem jako většina věcí, i tato modifikace má své úskalí. V tomto případě se jedná o velkou paměťovou náročnost a rychlý přístup k nalezeným tzv. *lokálním optim* (jelikož se jedná o optimální řešení triangulace polygonu, mohou se tyto dílčí výsledky takto nazývat).
Proč velká paměťová náročnost? Je třeba si uvědomit, že v každém uzlu stromu mohou být získány různé hraniční polygony a jelikož se nedá předem určit, které budou v dalším prohledávání stromu potřeba, jeví se jako nejlepší pamatovat si vše, co jsme schopni.
Tím se dostává k druhému problému (nevnímaje omezení velikostí dostupné paměti), a to k rychlému zjištění, zda pro nějaký konkrétní polygon bylo již tzv. lokální optimum nalezeno.
Jak tyto problémy jsou řešeny se lze dočíst v následující kapitole. Ale ještě než bude tomuto věnována pozornost, vraťme se ještě na počátek této kapitoly – především k problému, když dostaneme neúplnou triangulaci, ve které oblast Ω_{empty} není spojitá. Příklad, jak taková to situace může vypadat je znázorněn na následujícím obrázku.



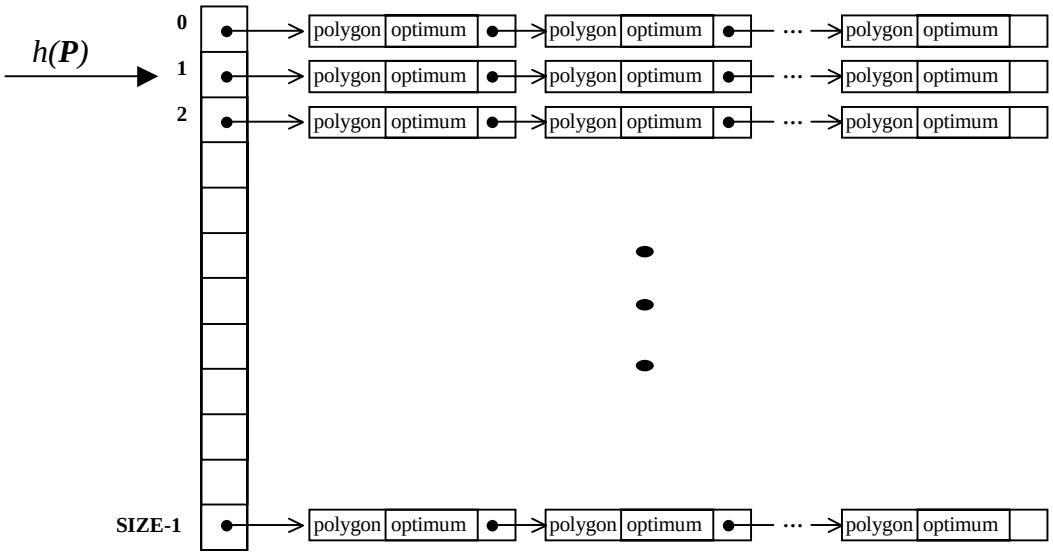
Obr. 3.4 Ukázka situací, kdy není oblast Ω_{empty} spojitá

Pokud se vrátíme k metodě vkládání trojúhelníků, lze zjistit, že již byl tento problém řešen (viz [Hla01]). Rozdělení oblasti Ω_{empty} je ekvivalentní rozdělení hraničního polygonu na více nespojitých polygonů. V původní metodě se tato situace řeší tak, že se nepracuje s jedním ale s množinou hraničních polygonů. Pokud tato množina není prázdná, je jisté, že ještě nebyla vytvořena úplná triangulace. Stejným způsobem lze postupovat i zde. Není třeba snažit se jedním krokem hnedka vytvořit výslednou triangulaci, ale může být použito tolik kroků kolik je nespojitých oblastí Ω_{empty} (samozřejmě za předpokladu, že již byla nalezena lok. optima pro jednotlivé polygony ohraničující této oblasti – pokud ne, je třeba vygenerovat potřebný podstrom).

4. Použití hash tabulky

Jak již bylo v závěru předchozí kapitoly zmíněno, je třeba navrhnout systém k uchování a rychlému přístupu k nalezeným lokálních optim pro různé hraniční polygony. Těch může být při procházení generovaným stromem velmi mnoho – záleží pouze na počtu bodů a jejich topologickém rozložení. Zde byla využita tzv. hash tabulka. Hlavním důvodem, proč byl použit právě tento systém, bylo především pro jednoduchost vkládání resp. odebírání dat, a i možnost rychlého vyhledávání.

Základním bodem tohoto systému je především použít vhodnou funkci *h(x)* (tzv. *hash funkci*), která by určovala hodnotu klíče do dané hash tabulky. Příklad takovéto hash tabulky je na následujícím obrázku.



Obr. 4.1 Hash tabulka

Jak je z obrázku 4.1 patrné, více různých polygonů může nabývat stejné hodnoty hash funkce. Tyto polygony jsou pak zřetězeny (jednosměrným seznamem) do tzv. clusterů. Počet těchto clusterů je roven délce hash tabulky (v tomto případě je rovna konstantě *SIZE*). Pro co nejrychlejší přístup k jednotlivým datům uložených v jednotlivých clusterech a optimální využití hash tabulky, měli by být zvolena hash funkce, která by zajišťovala následující podmínky:

- Délka clusteru musí být co nejkratší.
- Je využit celý rozsah hash tabulky pro uložení hodnot.

Při volbě funkce nesmí být opominuto, co bude použito jako vstupní hodnota pro danou funkci. V tomto případě se dá předpokládat, že to bude množina hran (přesněji indexy těchto hran), které tvoří hraniční polygon **P.**
Nalezení takovéto funkce je velmi obtížný úkol. Především proto, že pro obecné rozložení bodů nelze předem určit, jakých hodnot bude nabývat vstupní množina hran hash funkce (tj. hraniční polygon). Přesto byl navržen algoritmus, který celkem dobře (jak bude ukázáno v následující kapitole) splňuje výše uvedené předpoklady. Tento daný algoritmus by se dal zapsat následovně (viz obr. 4.2):

```
int hash ( unsigned int *bin_P, int numP ) {

      unsigned int index_hash = 111 * numP;
      for (i = 0; i < numblocks; i++) {
            index_hash = _rotl(index_hash,1);
            index_hash += bin_P;
      }
      unsigned int tmp_hash = index_hash;
      for (i = K; i < 32; i += K)
            tmp_hash ^= index_hash >> i;
      return (tmp_hash & MASK_VAL);
}
```

Obr. 4.2 Výpočet hash funkce

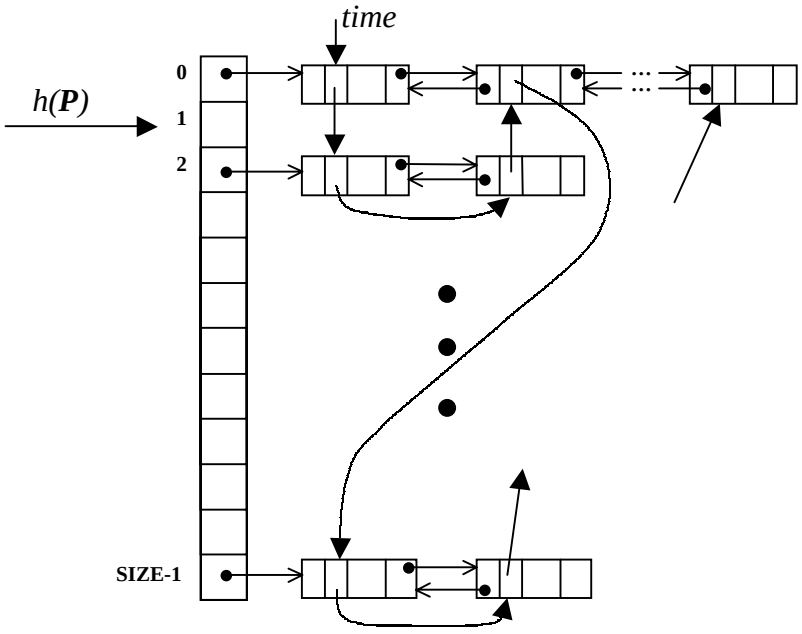
Význam použitých konstant a proměnných v algoritmu je následující:

bin_P	binární vektor reprezentující vstupní polygon (bit má hodnotu jedna, pokud hrana s příslušným indexem je součástí polygonu)
numP	počet hran polygonu
numblocks	počet bloků potřebných pro bin. reprezentaci polygonu (jako dat. typ reprezentující blok je použit 32 bit datový typ, tj, velikost jednoho bloku je 4 bytes)
K	udává mocninu pro výpočet délky hash tabulky – hash tabulka MUSÍ být rovna 2^K
MASK_VAL	hodnota rovna $2^K - 1$, použito pro převod výsledné hodnoty hash funkce do příslušného rozsahu
_rotl	operace rotace doleva o n bitů (s přenosem)
>>	operace posunu o n bitů doprava (doplněno z leva nulami)
^	operace XOR (exclusive OR)

Samotný princip celého algoritmu je velice jednoduchý. Nejprve se v prvním cyklu vezmou jednotlivé bloky a na základě operace rotace a aritmetického sčítání se vypočte hodnota v rozsahu typu *unsigned int* (tj. neznaménkové 32 bitové celé číslo). V druhém cyklu se pak toto číslo rozdělí na základě velikosti hash tabulky do bloků po *K* bitech a mezi těmito bloky se provede operace XOR. Výsledná hodnota je poté vrácena jako hodnota hash funkce. Aby bylo možné provést jednoduché rozdělení do bloků po *K* bitech před provedením operace XOR, je třeba, aby velikost hash tabulky byla rovna mocnině 2^K. Pokud by tato podmínka nebyla dodržena, bylo by třeba provést složitější operace než pouhé bitové posuny a maskování pro získání výsledné hodnoty v rozsahu velikosti hash tabulky.

5. Využití dostupné paměti počítače

I kdyby byla výše uvedená hash funkce optimální, je třeba upozornit ještě na jeden dosti závažný problém a to velikost dostupné paměti. Přestože v dnešní době lze na 32 bitových platformách dosáhnout adresace až 2GB (virtuální) paměti, nezaručuje to, že je to dostatek pro uchování celé hash tabulky. Počet generovaných uzlů (a tedy i nalezených lokálních optim) může být opravdu neomezeně mnoho, záleží pouze na počtu a topologickém rozložení bodů.
Tudíž je nasnadě hledat nějaký mechanismus, který by určoval, které lokální minima mohou být již zapomenuty a které si nadále pamatovat. Bohužel toto nelze pro obecné množiny bodů předem určit. Proto zde byl použit jednoduchý mechanismus založený na chronologickém pořadí nalezení jednotlivých optim, který se osvědčil (viz 6. kapitola). Vždy, když je potřeba uvolnit paměť, je odstraněno „nejstarší“ nalezené lokální minimum.
Prakticky toto pouze znamená, že se nad danými prvky clusterů vytváří pomocný jednocestný lineární seznam, kde se na konec přidávají nově vzniklé prvky a při výběru prvku, který se má zrušit se vybere vždy počáteční (viz následující obrázek).

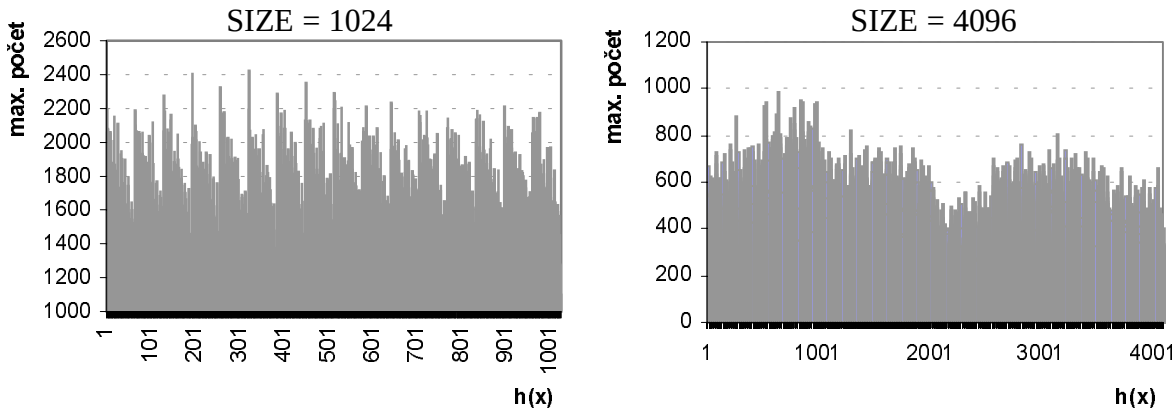


Obr.5.1 Hash tabulka s pomocným seznamem udávající chronologické pořadí prvků

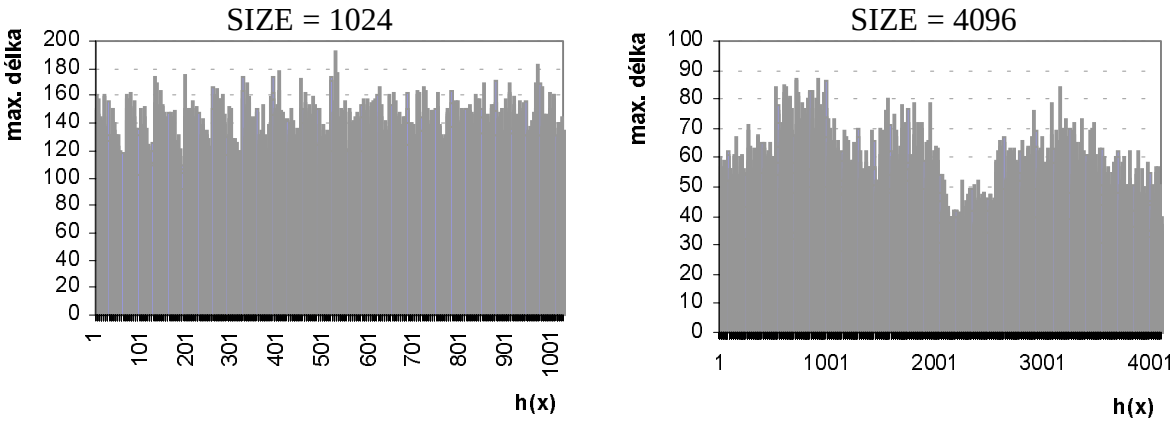
Na obrázku je také naznačeno, že již není pro zřetězení prvků v clusteru použit pouze jednocestný lineární seznam ale dvoucestný. Toto je pouze z důvodu snadnějšího vyloučení libovolného prvku z clusteru – každý prvek zná svého předchůdce i následníka.

6. Zhodnocení metody

Pro otestování této modifikované verze metody vkládání trojúhelníků, bylo (stejně jako u původní metody) použito jako ohodnocující kritérium nalezení triangulace s nejkratším součtem délek hran (Minimum Weight Triangulation). Je třeba také říci, že veškeré časy, které jsou v následujících grafech uvedeny, byly naměřeny na počítači firmy DELL, 2x650MHz, 1GB RAM, Windows 2000.
Ale nyní již přistupme k zhodnocení celé této metody. Prvním co bude především zajímavé, je ohodnocení zvolené hash funkce. Pokud by tato funkce nesplňovala požadované kritéria (viz 4. kapitola), urychlení, které jsme schopni dosáhnout zapamatování lok. optim, by bylo dosti zkreslené.
Při hodnocení kvality hash funkce by měl být především kladen důraz na rovnoměrné rozmístění dat ukládaných do hash tabulky a také na využití celé délky tabulky. Pro posouzení obou těchto faktorů jsem odzkoušel uvedenou hash funkci na desítkách datových množin a charakter této funkce si lze prohlédnout na následujícím obrázku.



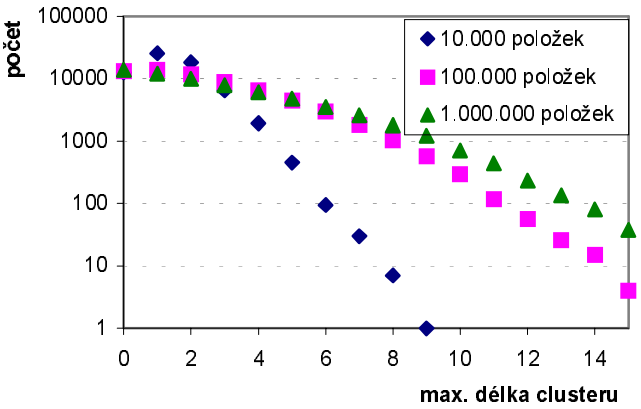
a) celkový počet zapsaných lok. optim do jednotlivých clusterů



b) maximální délky jednotlivých clusterů

Obr. 6.1 Grafy znázorňující charakter hash funkce při změně délky hash tabulky (náhodně vybraná množina 18 bodů)

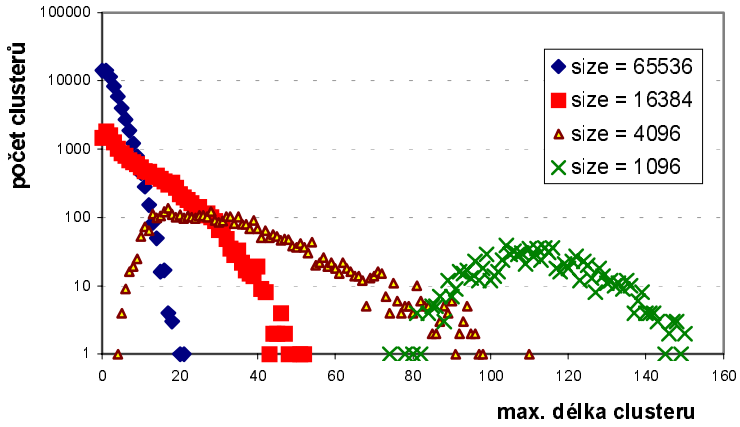
Na obrázku jsou znázorněny průběhy, které zachycují, jak se projeví změna délky hash tabulky pro danou množinu bodů. Horní graf ukazuje celkový počet ukládaných položek na daný klíč hash tabulky (cluster) během generování stromu. Ve spodním grafu jsou znázorněny maximální délky jednotlivých clusterů, kterých bylo během výpočtu dosaženo.
Přestože jsou na obrázku 6.1 uvedeny pouze výsledky jedné datové množiny, na základě porovnání s dalšími výsledky může být konstatováno, že charakter této hash funkce je pro všechny obdobný.
Na základě těchto informací lze o použité hash funkci celkově říci, že splňuje podmínku pro využití celé délky hash tabulky (což je potvrzeno i na obrázku 6.1). Bohužel rozložení dat do jednotlivých clusterů již není ideální. Ovšem díky tomu, že se pracuje s obecnou množinou bodů, lze těžko této vlastnosti dosáhnout. Tedy přestože po sobě následující délky clusterů mají spíše zubovitý charakter, může být prohlášeno, že zvolená hash funkce je pro dané účely postačující.
Zároveň může být porovnáním horního a spodního grafu zhodnoceno použití systému pro „zapomínání“ nalezených lok. optim, který je založen na vylučování prvků v chronologickém pořadí (viz 5. kapitola). Horní graf na obrázku vystihuje, jaké maximální délky by mohl cluster dosáhnout, pokud by se celá hash tabulka vešla do paměti, spodní graf zachycuje skutečnost. Jejich porovnáním lze zjistit, že mají celkem stejný charakter. Může být tedy konstatováno, že systém pro uvolnění paměti založený na vylučování prvků v chronologickém pořadí není také překážkou.
Dalším měřítkem, které by mohlo sloužit pro posouzení kvality hash funkce je histogram zachycující počet clusterů s danou max. délkou. Pro rychlou práci s hash tabulkou je potřeba, aby počet clusterů s velkou délkou byl pokud možno co nejmenší. Příklad znázorňující takovýto histogram pro danou množinu bodů při různém maximálním počtu lok. optim uložených v hash tabulce je znázorněn na následujícím obrázku (pro jiné množiny bodů byl obdobný).



Obr. 6.2 Graf znázorňující počet max. délek clusterů pro různé maxima hodnot uložených v hash tabulce (měřeno nad stejnou množiou 18 bodů, SIZE = 65536, celkový počet nalezených lok. optim = 533247)

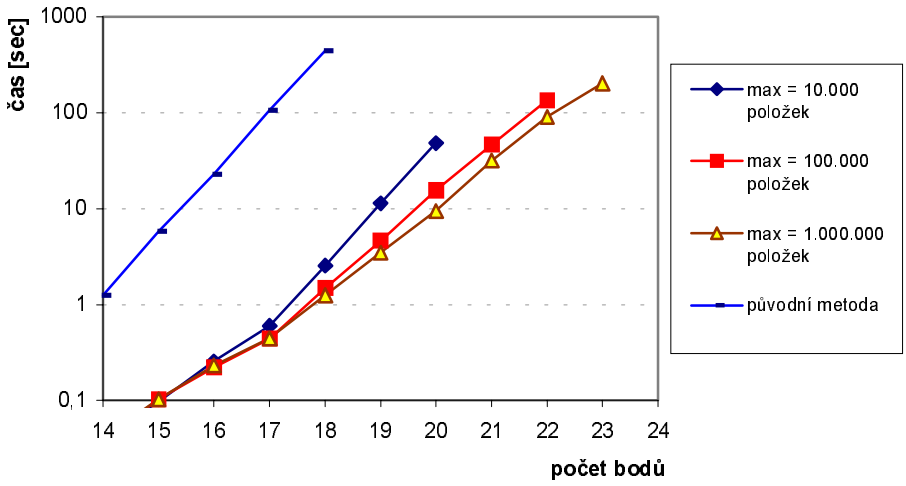
Z uvedeného grafu je patrné, že daná hash funkce má (obrazně řečeno) docela pěkné vlastnosti – s rostoucí délkou clusteru počet clusterů dané délky celkem rychle klesá. Také si lze povšimnout, že tabulka obsazuje řadu clusterů, které mají nulovou délku. To

je především způsobeno, dosti velkou délkou tabulky (SIZE = 2^{16} položek). Dalo by se říci, že zvětšování délky tabulky má v tomto smyslu negativní vliv, ale jak je na následujícím obrázku znázorněno, na druhou stranu to vede k celkovému snižování počtu dlouhých clusterů.



Obr. 6.3 Graf znázorňující změnu počtu clusterů se stejnou délkou při změně délky hash tabulky (stejná datová množina 18 bodů jako u obr. 6.2)

Shrne-li se vše, co bylo dosud zjištěno, lze říci, že čím větší bude délka hash tabulky, tím bude menší délka clusterů a tím bude rychlejší vyhledávání uložených hodnot v jednotlivých clusterech. Ovšem s rostoucí délkou hash tabulky nám roste i paměťová náročnost pro její vytvoření. Je tedy vhodné zvolit nějaký kompromis, mezi těmito dvěma faktory.
V tomto případě je hash tabulka rozdělena do 2^{16} clusterů, tj. tabulku tvoří 65536 clusterů. Tato hodnota byla zvolena jednak s ohledem na rozsahy dostupných datových typů a velikost potřebné paměti pro vytvoření prázdné hash tabulky (řádově megabyty).
Po posouzení kvality hash funkce a přibližném odhadu nastavení potřebných koeficientů, lze již přistoupit nejspíš k nejdůležitější věci, a to posouzení této vylepšené verze metody vkládání trojúhelníků z časového hlediska.
Jak již bylo v úvodu řečeno, doba potřebná pro nalezení výsledku je závislá na řadě faktorů. Z hlediska zjišťování závislosti na velikosti vstupní množiny bodů, je nejnepříjemnější závislost na topologickém rozložení jednotlivých bodů. Díky této závislosti, potřebný čas pro nalezení výsledků pro stejně velkou vstupní množinu, může být dosti odlišný. Proto je třeba brát uvedené časové hodnoty jako orientační.
Výsledné časové charakteristiky v závislosti na velikosti datové množiny jsou zobrazeny na následujícím obrázku 6.4.



Obr. 6.4 Doba potřebná pro nalezení MWT v závislosti na počtu bodů. Uveden původní algoritmus a modifikovaná metoda s různou max. hranicí dostupné paměti pro hash tab.

Jak již bylo v předchozím odstavci uvedeno, doba potřebná pro nalezení různých datových množin může být dosti odlišná. Aby tento problém byl pokud možno minimalizován, je v grafech uvedena průměrná hodnota časů získaná z několika měření stejně velkých datových množin (tj. množiny se stejným počtem bodů). Z grafu je i patrné, jak se podařilo díky pamatování si nalezených lokálních optim urychlit původní algoritmus metody vkládání algoritmů. Je zde uvedeno hnedka několik průběhů, které znázorňují, jak se bude měnit výsledná časová křivka se zvyšující se paměťovou náročností. Uvedené hodnoty *max* v grafu udávají kolik může být maximálně uloženo lok. hodnot v hash tabulce (viz 5. kapitola).

7. Závěr

Z uvedených výsledků je zřejmé, že využitím paměti, kterou je nám schopen počítač v současné době nabídnout, lze posunout hranici nalezení triangulace zadaných vlastností brutální silou o něco výše než u původní metody *Vkládání trojúhelníků* (viz [Hla01]). Přesto je třeba upozornit, že dosažené urychlení neřeší daný problém. Stále se jedná o nepolynomiální časovou složitost.
Další možnostmi urychlení algoritmu je využití paralelismu, resp. distribuce výpočtu viz [Hla01] a příloha A. Bylo ukázáno, že touto technikou nelze překonat řešení problému. Lze pouze posunout již zmiňovanou hranici maximálního počtu bodů o něco dál (řádově jednotky bodů).
Poslední možností urychlení algoritmů je využití některých matematických formulí, které byly pro dané kritérium dokázány. Většinou vedou k nalezení části hledané

triangulace pomocí algoritmu s polynomiální časovou složitostí. Ovšem zbytek je opět třeba dopočítat brutální silou, tj. použít algoritmus s nepolynomiální časovou složitostí.
Pro MWT existuje řada těchto matematických formulací. V [Hla01] je uveden vliv *NNG* a *Diamond* testu na rychlost výpočtu. Získané výsledky ukazují, že pokud daný test není dostatečně dobrý a nenalezne většinu hledané triangulace, ovlivní výsledný čas pouze minimálně. Např. pomocí uvedených testů se nám podařilo posunout hranici řádově o několik bodů. Z kategorie velmi dobrých testů lze například uvést LMT skeleton (viz [Bei97], [Dic96]), pomocí kterého (současně s modifikovanou metodou vkládání trojúhelníků) lze nalézt MWT řádově 100 až 150 bodů (viz příloha D), kde ale záleží na topologickém rozmístění bodů v rovině.
Závěrem lze říci, že sice lze problém nalezení triangulace podle obecného kritéria řešit, ale bohužel pro celkem malé množiny bodů. Pokud bychom chtěli získat lepší výsledky je třeba vědět o daném kritérium více informací, které by se dali využít pro zmenšení vstupní množiny algoritmů. Velmi dobrým příkladem nám může být výše uvedené MWT, kde jsme se díky LMT skeletonu dostali z původních řádově 20-30 bodů na hranici 100-150 bodů.
Nicméně na základě provedených experimentů lze konstatovat, že po použití hashovací funkce a LMT skeletonu pro MWT kritérium, lze dosáhnout generováním trojúhelníkových sítí brutální silou pro N v rozsahu cca 100-150 bodů v přijatelném čase cca jednotky až desítky hodin.

8. Literatura

[Bei97] Beiruti R.: A fast heuristic for finding the Minimum Weight Triangulation, Master thesis, Univeristé de Montréal, July 1997

[Dic96] Dickerson T. M., Montague H. M.: A (usualy?) connected subgraph of the Minimum Weight Triangulation. Proc. 12th Annu. ACM Symposium Computer Geometry, 1996

[Hla01] Hlavatý T., supervisor Skala V.: Generátor trojúhelníkových sítí zadaných vlastností brutální silou v E^2. Diplomová práce, Západočeská univerzita, Plzeň, květen, 2001

[Kol01] Kolingerová I., Ferko A.: Multicriteria-optimized triangulations. The Visual Computer, Vol. 17, No. 6, 2001, pp.380-395

Příloha A – Dodatek

A1. Metoda vkládání trojúhelníků

Tato kapitola obsahuje doplňující informace k metodě vkládání trojúhelníků, která byla popsána v kapitole 3.5 práce [Hla01]. V první části kapitoly je rozebráno, jaký vliv bude mít použití předzpracování (Diamond test a NNG test) na rychlost nalezení MWT (tj. triangulace s minimální součtem délek hran). Druhá část je zaměřena na možnost využití distribuovaných systémů pro urychlení nalezení MWT pomocí příslušné metody.

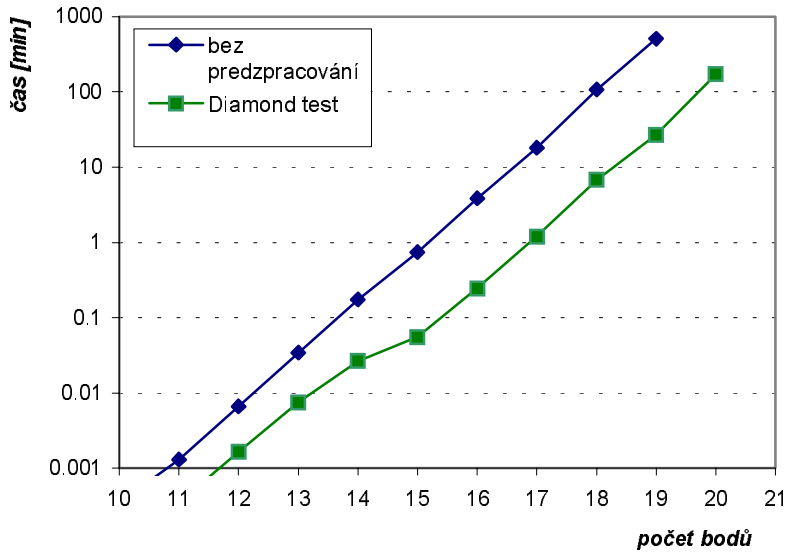
A1.1. Použití urychlovacích technik pro nalezení MWT

Ve čtvrté kapitole diplomové práce [Hla01] (dále jen DP nebo diplomová práce) byly popsány dvě předzpracování, které lze použít pro urychlení nalezení MWT. Jedná se o tzv. Diamond test a NNG test. Výsledky vlivu jednotlivých testů na rychlost výpočtu byly původně odzkoušeny na metodě odebírání hran z úplného grafu a jsou uvedeny v téže kapitole.
Tyto předzpracování lze samozřejmě použít i na zbylé metody. V této kapitole je popsáno, jakým způsobem bude ovlivněna rychlost výpočtu při použití těchto testů na tzv. metodu vkládání trojúhelníků (viz kapitola 3.5 DP).

A1.1.1. Diamond test

Jak již bylo dříve řečeno (viz kapitola 4.1 DP), pomocí Diamond testu lze najít některé hrany, u kterých je zaručeno, že nemohou být součástí hledané MWT. Při použití tohoto testu, u metody odebírání hran z úplného grafu, byly tyto hrany odebrány hnedka na začátku, takže se s nimi již nepracovalo. Dále také byly ze zbylé množiny hran odebrány i hrany, které se již neprotínali s žádnou jinou hranou. Ty naopak musí být součástí hledané MWT. Diamond test tedy měl hnedka dvojí využití. Nejdříve snížil vstupní množinu hran o hrany, které nemohou být v MWT, a potom ze zbylých hran byly ještě vyloučeny hrany, které naopak do MWT patří.
Při použití tohoto testu v metodě vkládání trojúhelníků, je třeba si nejdříve uvědomit, že se již zde nenahlíží na triangulaci jako na množinu hran, ale jako na množinu trojúhelníků. Základním elementem v této metodě již není hrana, ale tzv. prázdný trojúhelník (dále jen trojúhelník).
Vstupní množinou algoritmu je tedy množina trojúhelníků. Bude-li zjištěno, že nějaká hrana nemůže být součástí hledané triangulace, mohou být vyloučeny ze vstupní množiny všechny trojúhelníky, které danou hranu obsahují. Ty se už v hledané triangulaci objevit nemohou.

V opačné situaci, kdy se podaří najít hranu, která naopak je součástí hledané triangulace, není využití této informace tak jednoduché jako v předchozím případě. Aby se dalo prohlásit o nějakém trojúhelníku, že bude vždy součástí hledané triangulace, musí být všechny tři hrany, které tvoří trojúhelník součástí hledané triangulace. I přesto, že by se podařilo najít nějaký takovýto trojúhelník, jeho vyloučení ze vstupní množiny algoritmu a prohlášení o něm, že patří do MWT není možné.
V algoritmu metody vkládání trojúhelníků se pracuje s polygonem, který tvoří tzv. neztrianulovanou oblast (viz popis algoritmu – kapitola 2.5 DP). Pokud bychom tyto trojúhelníky vyloučily ze vstupní množiny, museli bychom neustále kontrolovat, zda výše uvedený polygon náhodou nenarazil na nějaký takovýto trojúhelník a případně polygon příslušně modifikovat. To by samozřejmě vedlo k značnému zvýšení složitosti algoritmu.
Souhrnem lze tedy říci, že pro urychlení metody vkládání trojúhelníku je možné pouze využít informace o hranách, které nabudou součástí hledané triangulace. Ze vstupní množiny algoritmu jsou vyloučeny všechny trojúhelníky, které nějakou takovouto hranu obsahují Tím se sníží vstupní množina algoritmu, což samozřejmě vede k urychlení.
Při ověření, jak velkého urychlení se podaří dosáhnout použitím tohoto předzpracování, bylo postupováno stejně jako ve všech předchozích případech (viz úvod třetí kapitoly DP, strana 36). Bylo vytvořeno několik počátečních množin s minimálním počtem bodů. Souřadnice bodů byly vygenerovány pomocí generátoru náhodných čísel s rovnoměrným rozdělením. Pro tyto množiny se změřil čas potřebný pro nalezení MWT příslušnou metodou a z výsledných hodnot se určila průměrná hodnota. Poté byl do daných vstupních množin přidán další náhodně vygenerovaný bod a celý postup se opakoval.
Výslednou časovou charakteristiku si lze prohlédnout na následujícím obrázku A1.1.1. Pro testování byl použit počítač DELL, PentiumIII, 2x450MHz, 1GB RAM.

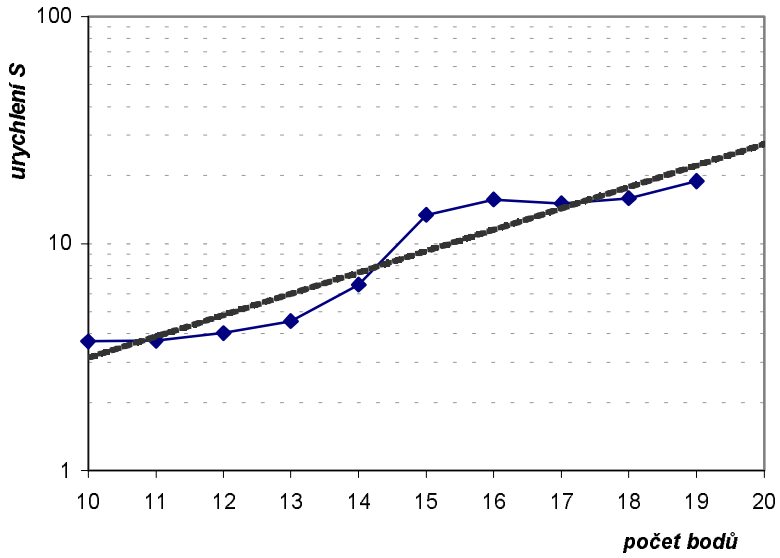


Obr. A1.1.1 Závislost doby výpočtu nalezení MWT metodou vkládání trojúhelníku na velikosti vstupní množiny bodů při použití Diamond testu v předzpracování ($\alpha = \pi/4,6$)

Z uvedených charakteristik se dá odhadnout, že se podařilo urychlit metodu vkládání trojúhelníků pro množiny 15 až 20 bodů řádově tak 10krát. Výrazné zlomy v průběhu křivky jsou především způsobeny počtem vyloučených hran pomocí Diamond testu, který je závislý na topologickém rozložení bodů.
Přesnější charakteristika průběhu urychlení v závislosti na rozložení bodů je znázorněna na následujícím obrázku (viz obr. A1.1.2.). Koeficient urychlení je počítán podle následujícího vzorce:

$$S = \frac{t_{bf}}{t_{diam}} \qquad \text{(A1.1.1)}$$

kde t_{bf} je čas potřebný pro výpočet bez použit Diamond testu a t_{diam} je čas, který je potřebný pro výpočet, pokud bude Diamond test použit.

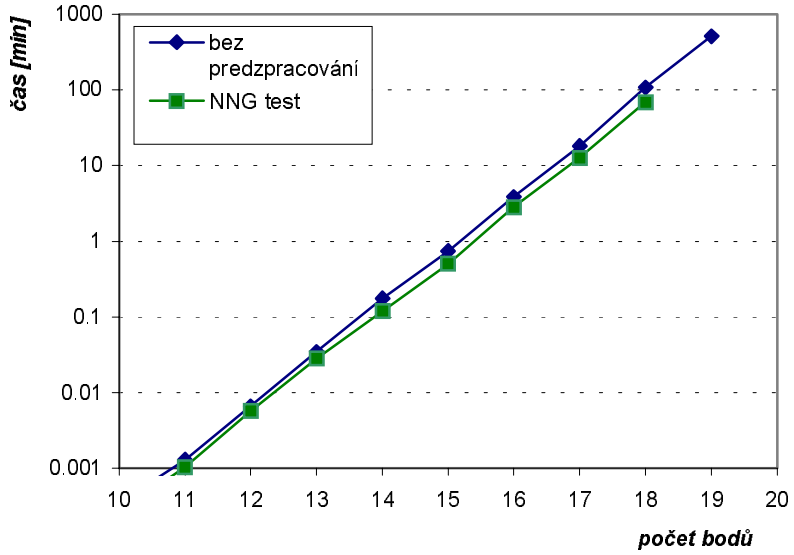


Obr. A1.1.2 Charakteristika urychlení metody vkládání trojúhelníků s použitím Diamond testu ($\alpha = \pi/4,6$) v předzpracování v závislosti na počtu bodů

Bohužel z charakteristiky průběhu křivky nelze mnoho vyčíst. Dá se pouze konstatovat, že křivka má s přibývajícím počtem bodů vstupní množiny vzestupný charakter. Přerušovanou čarou je v grafu naznačen průběh exponenciální funkce, která byla spočtena na základě aproximace daných hodnot. Zanedbá-li se odchylka hodnot od dané funkce, dalo by se říci, že urychlení má přibližně takovýto charakter. Ovšem pro přesnější odhad průběhu křivky by bylo třeba mnohem více hodnot, což bohužel z časových nároků potřebných pro výpočet není realizovatelné.

A1.1.2. NNG test

Další z uvedených předzpracování byl tzv. NNG test (podrobněji viz kapitola 5.2 v DP). Pomocí tohoto testu se dá najít některé hrany, které patří do MWT. Ovšem jak již bylo řečeno v předchozí kapitole zabývající se Diamond testem, využití znalosti hran patřících do hledané triangulace je u metody vkládání trojúhelníků problematické. Ovšem přesto lze i tento test použít pro urychlení algoritmu. Pokud je totiž nalezena nějaká hrana patřící do MWT, mohou být vyloučeny všechny hrany, které se s patřičnou hranou protínají. Tímto tedy dostaneme množinu hran, které opět v MWT být nemohou a můžeme stejným způsobem jako u Diamond testu snížit vstupní množinu trojúhelníků.
Pro ověření, jak velmi se ovlivní rychlost výpočtu při použití tohoto testu byl použit stejný postup jako v předchozím případě. Výslednou časovou charakteristiku (testováno na stejném počítači) si lze prohlédnout na následujícím obrázku.

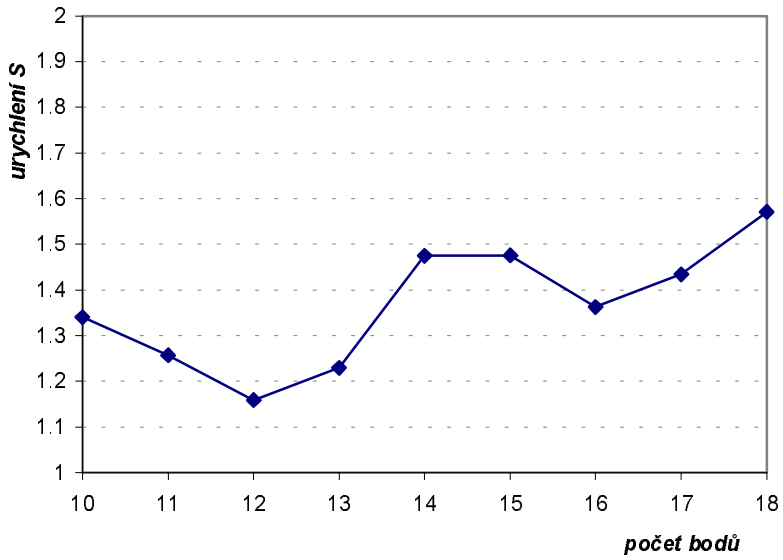


Obr. A1.1.3 Závislost doby výpočtu nalezení MWT metodou vkládání trojúhelníku na velikosti vstupní množiny bodů při použití NNG testu v předzpracování

Je patrné, že tento test neovlivňuje dobu potřebnou pro výpočet tak silně jako Diamond test. Ovšem přece jenom i zde bylo nějakého urychlení dosaženo. Vezme-li se v úvahu doba potřebná pro provedení NNG testu, která je pro takovýto malý počet zanedbatelná (složitost algoritmu je $O(N^2)$), dá se říci, že i zde došlo k určitému pokroku.
Přesnější průběh urychlovací křivky je znázorněn na obrázku A1.1.3. Urychlení je zde počítáno stejným způsobem jako u Diamond testu:

$$S = \frac{t_{bf}}{t_{NNG}} \qquad \text{(A1.1.2)}$$

t_{bf} je čas potřebný pro výpočet bez použití NNG testu a t_{NNG} je čas, který je potřebný pro výpočet, pokud bude NNG test použit.



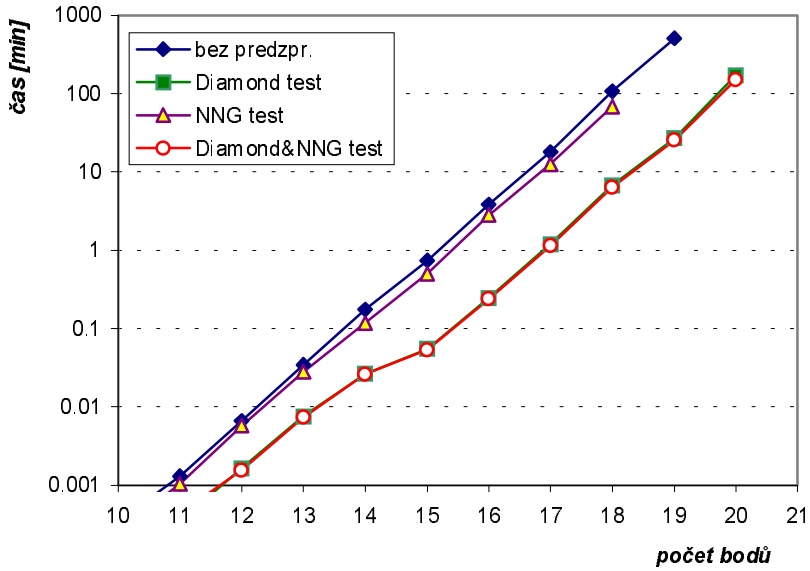
Obr. A1.1.4 Charakteristika urychlení metody vkládání trojúhelníků s použitím NNG testu v předzpracování v závislosti na počtu bodů

Z výsledného průběhu křivky se bohužel nedá mnoho říci. Je vidět že urychlení algoritmu není moc velké a charakter křivky lze těžko nějakým způsobem popsat. Zdá se, že s rostoucím počtem bodů křivka má průměrně vzestupný charakter. Pokles křivky v některých místech, je způsobem silnou závislostí počtu vyloučených hran NNG testem na topologickém rozložení bodů Tato závislost je rozhodně větší než u Diamond testu.

A1.1.3. Použití Diamond testu a NNG testu současně

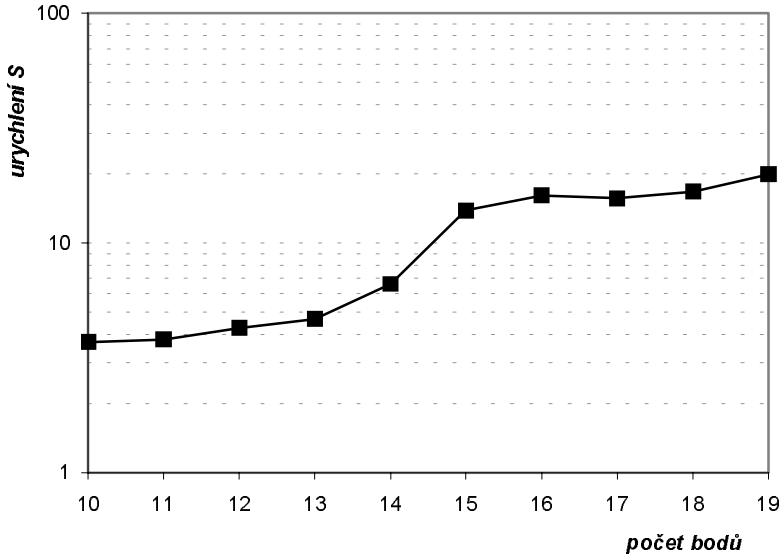
Cílem celé práce je především v reálném čase nalézt požadovanou triangulaci pokud možno pro co největší počet bodů. Je tedy výhodné použít obě předzpracování současně. Jejich časová složitost je polynomiální (Diamond test má složitost $O(N^3)$ a NNG test má složitost $O(N^2)$, kde N je počet bodů) a pro takto malé množiny bodů je v podstatě prakticky nulová.
O množinách vyloučených hran jednotlivými testy obecně platí, že jejich průnik nemusí být prázdná množina, ale zároveň se nedá říci ani to, že by jedna množina byla zároveň podmnožinou druhé. Výsledné urychlení se tedy bude pohybovat mezi urychlením, které je možné dosáhnout použitím Diamond testu (to se průměrně jeví větší než u NNG testu), a maximálním urychlením, které je rovno součtu urychlení jednotlivých testů použitých nezávisle na sobě.
Časový průběh rychlosti výpočtu při použití obou testů zároveň, je zobrazen na následujícím obrázku (viz obr. A1.1.5). Pro lepší porovnání, jsou do grafu přidány i časové průběhy naměřené jak bez použití předzpracování, tak i při použití pouze

jednoho testu. Výsledné časy byly naměřeny opět na stejném počítači (DELL, PentiumIII, 2x450MHz, 1GB RAM).



Obr. A1.1.5 Závislost doby výpočtu nalezení MWT metodou vkládání trojúhelníku na velikosti vstupní množiny bodů při použití předzpracování

Porovná-li se průběh při použití obou testů současně s časovým průběhem, který byl získán pouze použitím Diamond testu lze říci, že výsledných charakter křivky se o moc nezměnil. To je patrné i z gafu urychlení, který je zobrazen na obrázku A1.1.6, který se velmi podobná grafu urychlení pro Diamond test (viz obr. A1.1.2.).



Obr. A1.1.6 Charakteristika urychlení metody vkládání trojúhelníků s použitím Diamond testu ($\alpha = \pi/4{,}6$) a NNG testu v předzpracování v závislosti na počtu bodů

A1.2. Přibližný časový odhad časové náročnosti výpočtu

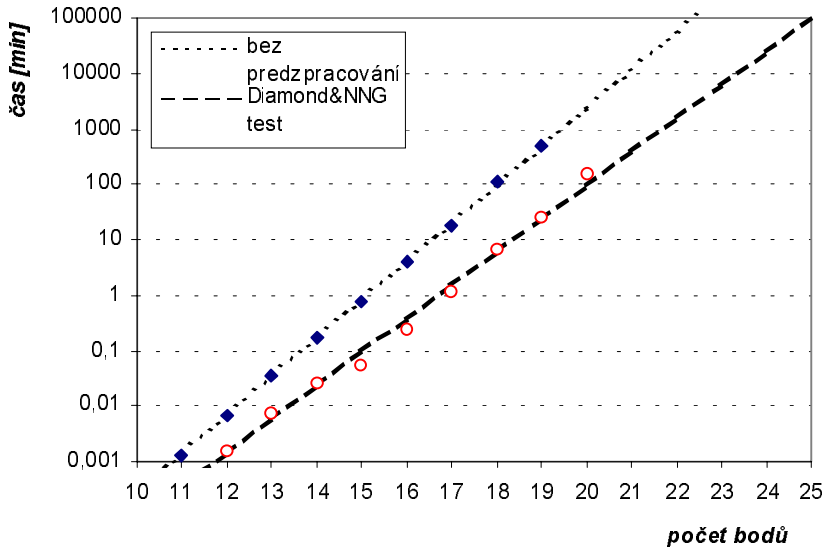
Křivky, vyjadřující časovou závislost doby potřebnou pro výpočet v závislosti na velikosti vstupní množiny bodů (viz obrázek A1.1.6), mají téměř exponenciální charakter. Na základě této skutečnosti by se dal z uvedených hodnot odvodit přibližný odhad doby potřebné pro výpočet pro rozsáhlejší množiny bodů. Výsledkem by měla být funkce v následujícím tvaru:

$$f_t(N) = a \cdot m^N \qquad \text{(A1.2.1)}$$

kde a a m jsou konstanty charakterizující průběh dané křivky a N je počet bodů.
Na základě aproximace hodnot, byly odvozeny následující rovnice:

$$t_{BF}(N) \doteq 3.72 \cdot 10^{-11} \cdot (4.894)^N$$
$$t_{PR}(N) \doteq 8.53 \cdot 10^{-11} \cdot (4.004)^N \qquad \text{(A1.2.2)}$$

kde t_{BF} je funkce odhadované časové náročnosti pro metodu bez použití předzpracování a t_{PR} je funkce odhadované časové náročnosti s použitím obou předzpracování (Diamond a NNG testu). Samozřejmě je nutné také říci, že tyto rovnice platí pouze pro počítač, na kterém byly časy naměřeny (DELL, PentiumIII, 2x450MHz, 1GB RAM).
Způsob jakým uvedené funkce aproximují naměřené hodnoty si lze prohlédnout na následujícím obrázku, kde průběhy jednotlivých funkcí jsou naznačeny přerušovanou čarou.



Obr. A1.2.1. Odhadovaný čas Závislost doby výpočtu nalezení MWT metodou vkládání trojúhelníku na velikosti vstupní množiny bodů při použití NNG testu v předzpracování

A2. Distribuce metody vkládání trojúhelníků

Další možností, jak urychlit dobu potřebnou pro výpočet, je využití distribuovaných sítí. V dnešní době je to dosti populární trend a již jsou k dispozici lokální počítačové sítě skládající se až ze stovek počítačů.
V páté kapitole DP byla tato možnost urychlení doby výpočtu popsána pro metodu odebírání hran z úplného grafu. Byly zde uvedeny i případné modifikace, které je třeba udělat pro zajištění správné funkčnosti programu v distribuovaném prostředí.
V následujícím textu bude provedena podobná analýza pro metodu vkládání trojúhelníku.

A2.1. Modifikace metody

Stejně jako tomu bylo u metody odebírání hran, je třeba rozdělit generovaný strom na části, které se potom budou moci přidělit jednotlivým počítačům tvořící síť, na kterou se bude výpočet distribuovat. Kdyby byla síť tvořena ze stejných typů počítačů a dala by se zanedbat i rychlost komunikace mezi jednotlivými počítači, stačilo by generovaný strom rozdělit na tolik stejně velkých podstromů, kolik je počítačů. Ovšem to obecně neplatí a i kdyby tomu tak bylo, nebylo by to prakticky možné, protože se nedá předem určit kolik uzlů bude generovaný strom mít.
Je tedy otázka, jakým způsobem generovaný strom rozdělit tak, aby byla zaručena dobrá efektivita výpočtu, tj. aby všechny počítače dopočítaly přibližně ve stejnou dobu. Tento problém je vyřešen velmi jednoduše.
Ještě před tím, než je výpočet spuštěn v distribuovaném prostředí, je na řídícím počítači vygenerována část stromu pomocí strategie prohledávání stromu do šířky (viz definice 2.10. v DP). Důvod zvolení této strategie je následující:
Na základě znalosti, že všechny triangulace jsou složeny ze stejného počtu trojúhelníků (viz věta 2.2. v DP), dá se předpokládat, že uzly, které se nacházejí ve stejné hloubce vytvářeného stromu, budou mít přibližně stejně rozsáhlé podstromy. Zpracování těchto uzlů by mělo být tedy přibližně stejně časově náročné.
Zmiňovaná strategie upřednostňuje při expanzi uzlů právě uzly s nejmenší hloubkou ve stromu, tj. dokud není provedena expanze všech uzlů v dané hloubce stromu, neexpandují se uzly s větší hloubkou. Podrobněji je celý algoritmus popsán v následující kapitole.

A2.2. Popis algoritmu

Jak již bylo řečeno, prvním krokem před distribuováním výpočtu na další počítače je vytvoření určitého počtu nezpracovaných uzlů pomocí strategie prohledávání stromu do šířky. Tyto uzly budou poté rozeslány mezi jednotlivé počítače, na kterých by měl výpočet probíhat. Zde se s nimi bude pracovat jako kdyby představovali kořenový uzel

generovaného stromu a budou podle toho náležitě zpracovány. Pro jejich zpracování může být použit původní verze algoritmu vkládání trojúhelníků, který byl popsán v kapitole 3.5 diplomové práce (viz obr. 3.5.2 v DP). Pouze jedinou změnou bude to, že kořenovým uzlem nebude již polygon reprezentující konvexní obálku, ale právě daný uzel, který má být zpracován.
Pro zajištění potřebné komunikace mezi jednotlivými počítači a distribuce dat, byl použit systém GSD (*General Systém for Distribution*), který je v současné době vyvíjen na Západočeské univerzitě (podrobnější informace o tomto systému lze najít v příloze C diplomové práce). Podstatě se jedná o systém, který paralelizuje úlohy ve smyslu *farmer-worker*, tj. je jeden řídící počítač, který rozděluje prácí svým „podřízeným" počítačům (dělníkům) a po dokončení práce, opět převezme patřičné výsledky. Celý systém poskytuje dvě základní funkce:

- zajištění komunikace mezi řídícím počítačem a počítači, na kterých bude probíhat výpočet

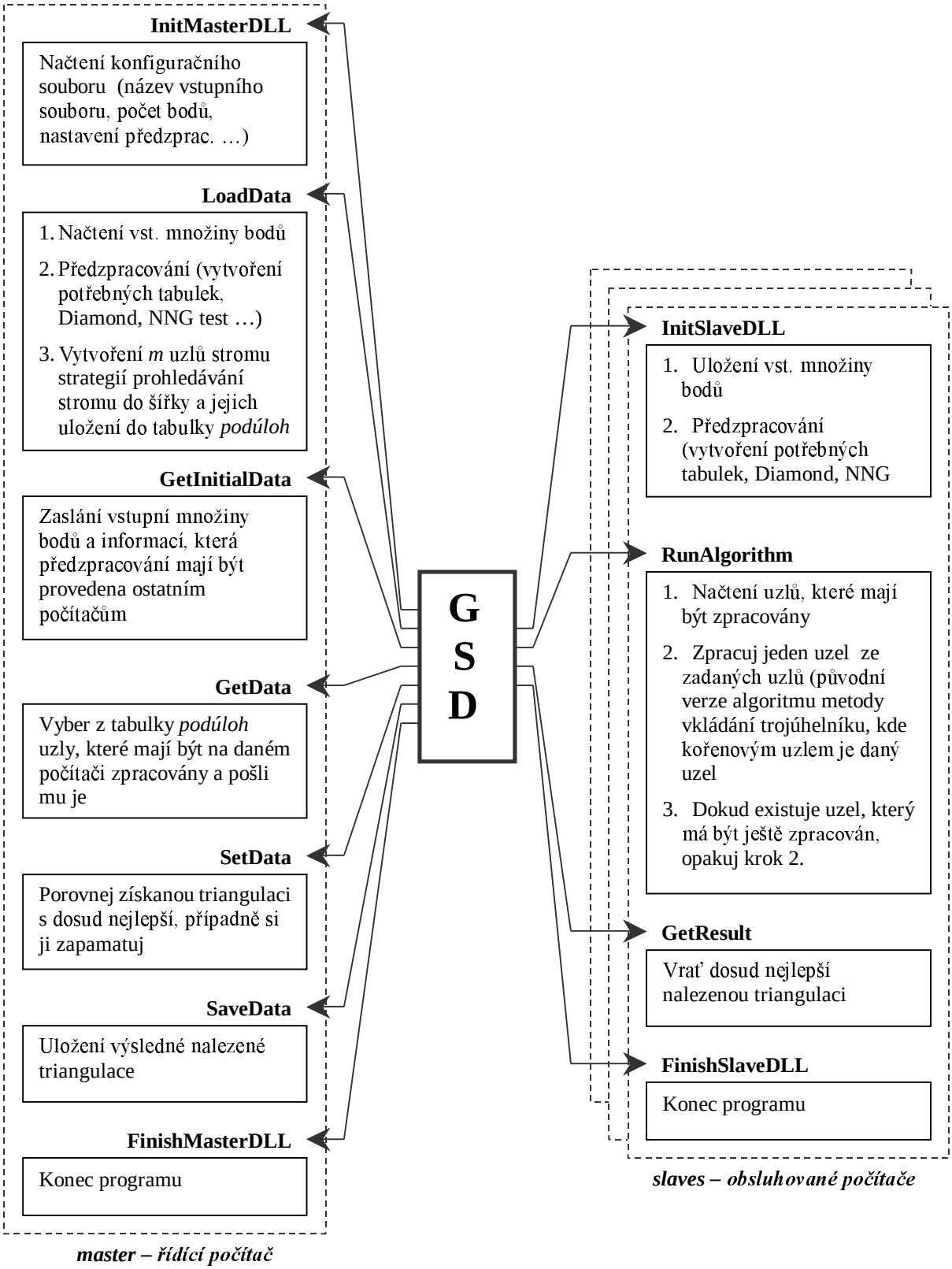
- zajištění efektivního rozdělení práce mezi jednotlivé počítače

V podstatě se jedná o nástroj, který umožňuje uživateli navrhnout algoritmus pro použití v distribuovaných systémech tak, aby se uživatel nemusel zabývat věcmi týkající se samotné distribuce výpočtu.
Úloha je rozdělena na dílčí podúlohy, které v tomto případě jsou reprezentovány uzly vytvořenými v sekvenční části algoritmu (strategií prohledávání stromu do šířky). Systém GSD potom zajistí rozeslání těchto dílčích podúloh ostatním počítačům, na kterých jsou patřičně zpracovány.
Celý program je v podstatě navržen jako několik funkcí, které jsou volány systémem GSD. Názvy i význam jednotlivých funkcí je pevně stanoven a jejich podrobný popis lze najít v příloze A na konci této práce. Hrubý náhled na rozvržení celého programu do jednotlivých funkcí si lze prohlédnout na obrázku A2.2.1.
Mezi nejdůležitější body implementace jednotlivých funkcí patří následující dvě části:

- Navržení algoritmu pro vytvoření tabulky podúloh pomocí strategie prohledávání uzlu do šířky (třetí bod ve funkci *LoadData*)

- Navržení algoritmu pro vygenerování stromu od vybraného uzlu s případným ohodnocením nalezených triangulací (druhý bod funkce *RunAlgorithm*)

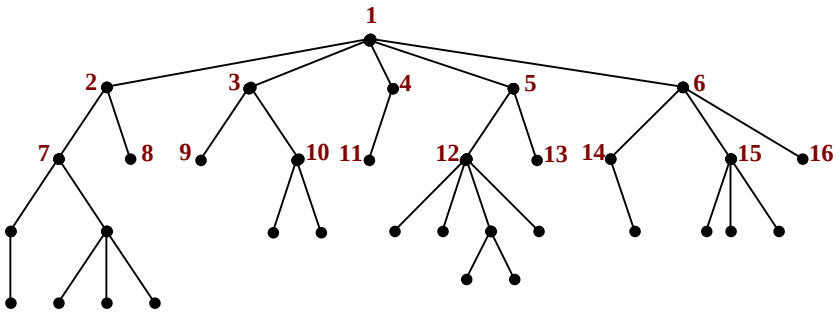
Zbylá část popisu bude proto zaměřena právě na tyto dvě části, které v podstatě tvoří jádro celého algoritmu.



Obr. A2.2.1 Základní popis funkcí jednotlivých procedur pro implementaci metody vkládání trojúhelníků v systémem GSD

A2.2.1. Algoritmus vytvoření tabulky úloh

Pod pojmem tabulka úloh je skryta tabulka, ve které jsou uloženy uzly stromu, které je třeba dále expandovat, aby byl vygenerován kompletně celý strom. Jak již bylo několikrát zmíněno tabulka úloh je vytvořena pomocí strategie procházení stromu do šířky. Hlavním důvodem volby této strategie bylo především, aby v tabulce byly uložené uzly, jejíchž další expanze by byla přibližně stejně časově náročná. Pokud je úloha rozdělena na přibližně stejně rozsáhlé podúlohy, lze lépe odhadnout, jakým způsobem zpracování těchto podúloh rozvrhnout mezi jednotlivé počítače, aby byly efektivně vytíženy po celou dobu výpočtu. Přidělování jednotlivých podúloh má na starosti samotný systém, není tedy důvod se tímto problémem dále zabývat. Jediný naším úkolem je pouze rozdělit strom na patřičný počet podstromů.
Na následujícím obrázku je zobrazen strom, kde u jednotlivých uzlů jsou přidělena čísla symbolizující pořadí, ve kterém je třeba provést expanzi uzlu, aby byla dodržena strategie prohledávání stromu do šířky.



Obr. A2.2.2 Příklad stromu s naznačením pořadí (pomocí očíslování některých uzlů), ve kterém je strom procházen při použití strategie prohledávání stromu do šířky

Ve výsledné tabulce úloh by měli být uloženy pouze uzly, které nebyly ještě expandovány. Je-li tedy provedena expanze daného uzlu, lze ho posléze zapomenut. Stačí si pamatovat pouze nově vzniklé stromy uzly.
Popis algoritmu, který vytváří výslednou tabulku úloh na základě uvedené strategie procházení stromu, je stručně uveden na obrázku A2.2.3. Vstupním bodem algoritmu je původní kořenový uzel stromu (tj. konvexní obálka zadané množiny bodů) a také požadovaná velikost tabulky, která má být vytvořena. Pokud tabulka bude obsahovat zadaný počet uzlů nebo dokonce bude překročen, expanze dalších uzlů se již neprovádí a algoritmus končí.

1. Vytvoření kořenového uzlu (tj. uzlu, ve kterém hraniční polygon tvoří hrany konvexní obálky) a uložení jej na první položku tabulky (index 0)

2. Nastavení m_{max} na hodnotu rovnou počtu uzlů, které je třeba vygenerovat zmenšenou o jedničku

3. Nastavení indexu na aktuální položku tabulky *i*, indexu na poslední obsazenou položku tabulky i_{max} a indexu na poslední položku tabulky s aktuální hloubkou stromu max_{deep} na hodnotu 0 (první položka tabulky)

4. Nastavení log. proměnné *existNew = FALSE* (udává, zda expanzí uzlů v dané hloubce stromu vznikly nové uzly, které by se dali dále expandovat)

5. Provedení expanze uzlu, na který odkazuje do tabulky index *i* a uložení nově vzniklých uzlů na konec tabulky

6. **IF** počet nově vzniklých uzlů je větší než nula **THEN**

    a. Přepsání v tabulce položku *i* poslední zapsanou položkou

    b. Zvýšení hodnoty i_{max} o počet nově vzniklých uzlů bez jedné

    c. *existNew = TRUE*

7. **IF** $i_{max} >= m_{max}$ **THEN**

    a. Skok na bod **10**

8. **IF** $i = max_{deep}$ **THEN**

    a. **IF** *existNew = FALSE* **THEN** Skok na bod **10**

    b. $i = 0,\ max_{deep} = i_{max},$ *existNew = FALSE*

    ELSE

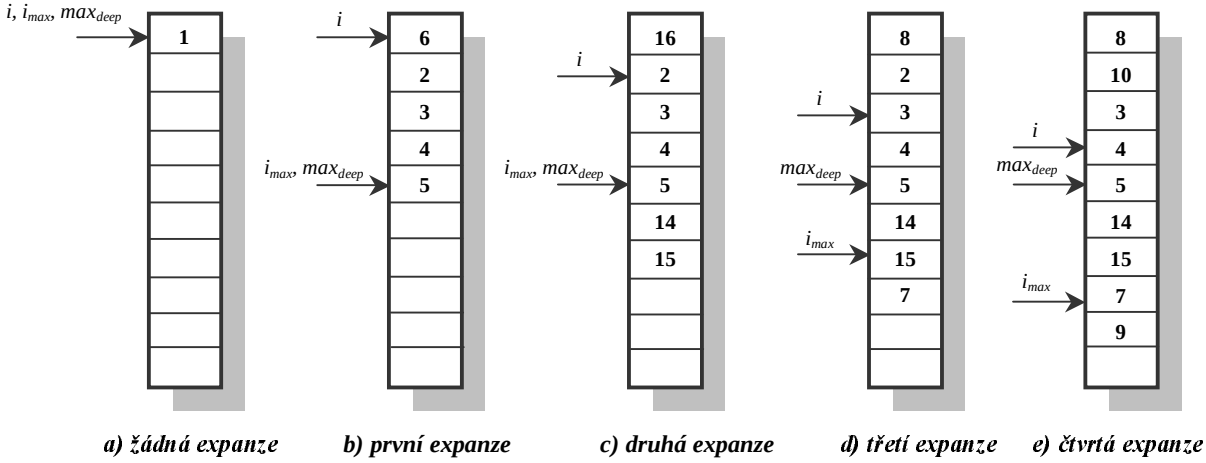
    a. zvýšení indexu *i* o jedničku

9. Skok na bod **5**

10. Konec algoritmu

Obr. A2.2.3 Popis algoritmu vytvoření tabulky úloh na základě strategie procházení stromu do šířky

Příklad způsobu fungování uvedeného algoritmus je demonstrován na následujícím obrázku (viz obr. A2.2.4), ve kterém je naznačen postup, jakým způsobem se vyplňují hodnoty v tabulce pro strom, který má stejnou strukturu jako je uvedeno na obrázku A A2.2.2. Uvedené hodnoty v jednotlivých položkách tabulky odpovídají číslům uzlů, které jim byla přidělena (viz obr. A2.2.2) a uvedené tabulky odpovídají stavu, ve kterém se nacházejí před provedením expanze daného uzlu (tj. před provedením kroku 5 podle daného algoritmu).



Obr. A2.2.4 Postup vyplnění hodnot v tabulce úloh pro strom zobrazený na obr. A2.2.2 (uvedená čísla v tabulce odpovídají číslům jednotlivým uzlům stromu)

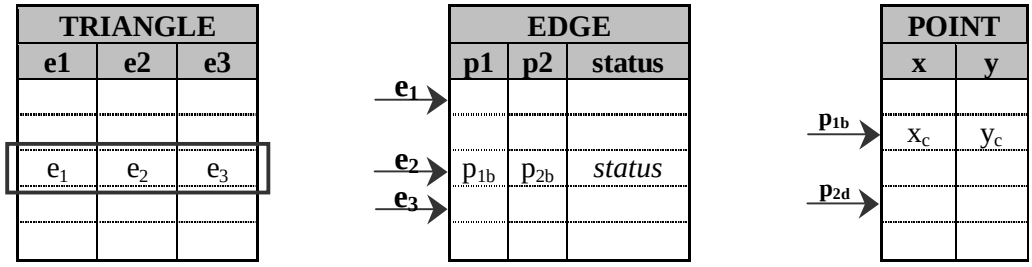
Dalo by se podotknout, že uvedený algoritmus nezpracovává uzle v přesném pořadí, jak je uvedeno na obrázku A2.2.2. To je způsobeno tím, že poslední uzel vzniklý expanzí, se přesouvá v tabulce na pozici uzlu původního. Ovšem základní princip strategie prohledávání stromu do šířky není tímto nějak narušen a tedy uvedený algoritmus je plně funkční.

Doposud se zde hovořilo obecně o uzlech stromu. Ovšem je nutné si uvědomit, že uzly generovaného stromu reprezentují tzv. *neúplné triangulace*, které jsou vytvářeny podle daného algoritmu, tj. metody vkládání trojúhelníků. Příklad, jak takovýto strom může vypadat, lze nalézt na obrázku 3.5.2 v DP. Aby mohla být provedena expanze daného uzlu (ať se jedná o strategii prohledávání stromu do šířky nebo do hloubky) je třeba zajistit, že v daném uzlu budou uloženy veškeré potřebné informace popisující příslušnou neúplnou triangulaci.

Ještě před tím, než zde bude popsána zvolená datová struktura popisující daný uzel, je třeba seznámit se s dalšími tabulkami, které se vytvářejí během předzpracování z důvodu usnadnění práce a samozřejmě i z důvodu urychlení algoritmu. Bez jejich pochopení si lze jen těžko představit význam jednotlivých položek struktury popisující uzel generovaného stromu.

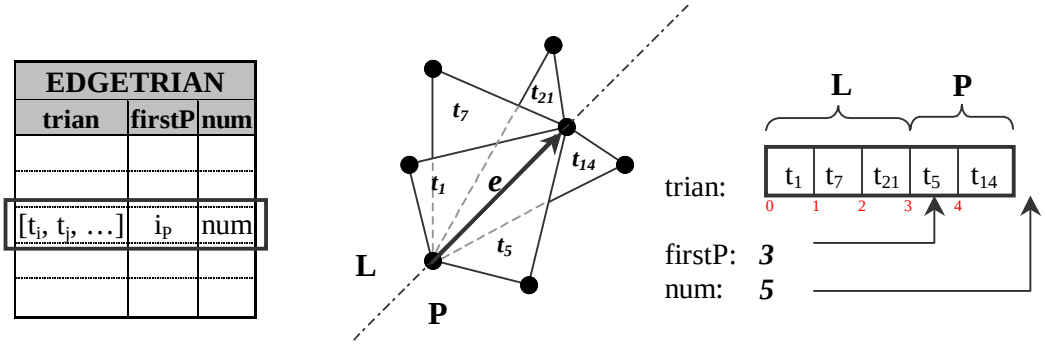
Jak je již u metody vkládání trojúhelníků zmíněno a jak i samotný název napovídá, na triangulaci je zde nahlíženo jako na množinu tzv. prázdných trojúhelníků. Pro jednodušší práci s danými trojúhelníky je tedy vhodné vytvořit si tabulku, která bude obsahovat informace popisující jednotlivé trojúhelníky. Tato tabulka je pojmenována

triangle a její struktura je zobrazena na obrázku A2.2.5. V podstatě každý trojúhelník je popsán třemi indexy na hrany tvořící příslušný trojúhelník. Tyto indexy se odkazují do tabulky *edge*, kde jsou zase uloženy informace o příslušných hranách.
Tyto informace tvoří především index (do tabulky *point*) počátečního a koncového bodu určující příslušnou hranu a status, ve kterém se zaznamenává informace o jaký typ hrany jde. Pomocí hodnoty v položce *status* lze například určit zda se jedná o hranu konvexní obálky, o hranu vyloučenou pomocí Diamond testu případně jiného předzpracování, atd.



Obr. A2.2.5 Naznačení struktury tabulek pro uchování informací o bodech, hranách a trojúhelnících

Další tabulkou, která je vytvořena, je tabulka *edgeTrian*. Počet položek v této tabulce je shodný s počtem hran a jejich obsah ukrývá seznam trojúhelníků, které přiléhají k dané hraně. Hodnota indexu do tabulky *edgeTrian* je totožná s hodnotou indexu do tabulky *edge*, čímž je tedy i řečeno, o jakou hranu se v tabulce jedná. Pro urychlení algoritmu byly trojúhelníky rozděleny ještě do dvou skupin a to na trojúhelníky přiléhající k hraně zleva (L) a trojúhelníky přiléhající k hraně zprava (P). Z jaké strany daný trojúhelník přiléhá, je určeno pomocí orientace hrany, která je směrována od bodu, který má nižší index v tabulce *point* k bodu s vyšším indexem. Struktura dané tabulky je naznačena na následujícím obrázku (viz obr. A2.2.6). Je zde uveden i jednoduchý příklad s nastavením jednotlivých položek hodnot v daném řádku tabulky.

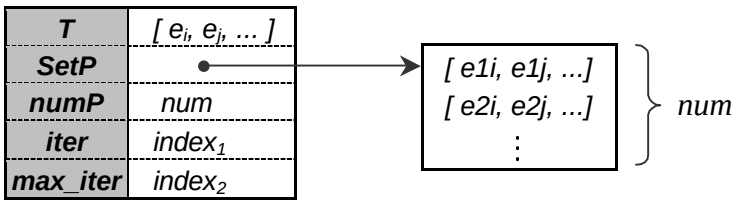


Obr. A2.2.6 Popis struktury tabulky edgeTrian, ve které jsou uloženy informace, které trojúhelníky přiléhají k příslušné hraně

Nyní je již možné vrátit se k popisu samotného uzlu stromu. V podstatě se s uzly vytvářeného uzlu bude pracovat stejně, jako tomu bylo u původní verze metody vkládání trojúhelníků. Pouze je změněna strategie, jakou je daný strom procházen. V uzlu by tedy měli být zaznamenány následující informace (v závorkách je uvedeno i zavedené označení pro dané položky):

- seznam hran, které již byly do triangulace vloženy (pole **T**)
- seznam množin hran tvořící hraniční polynomy (pole **setP**)
- počet množin polynomů (**numP**)
- index ukazující do tabulky *edgeTrian* určující, od kterého trojúhelníku bude zahájena expanze uzlu (**iter**)
- index ukazující do tabulky *edgeTrian* určující, u kterého trojúhelníku bude expanze uzlu skončena (**max_iter**)

Graficky jsou jednotlivé položky popisující uzel znázorněny na obrázku A2.2.7.

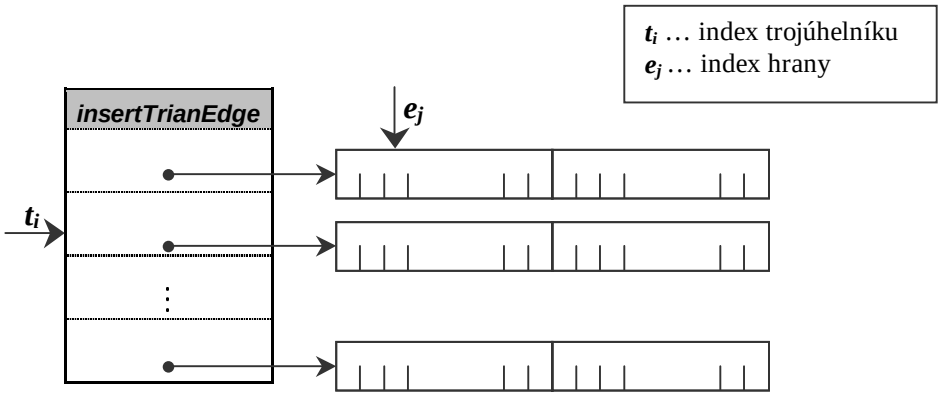


Obr. A2.2.7 Popis struktury položek pro uzel generovaného stromu

Poslední tabulkou, která se vytváří během předzpracování je tabulka *intersectTrianEdge*. Sice není potřebná pro popis uzlu stromu, ale její význam není o to menší. Jedná se v podstatě o dvou rozměrnou tabulku, ve které je uložena informace, s kterými hranami se každý trojúhelník uložený v tabulce *triangle* protíná.
Index řádku tabulky určuje daný trojúhelník a je totožný s hodnotou indexu do tabulky *triangle*. Sloupcový index naopak určuje hranu a je totožný s indexem do tabulky *edge*. Jelikož takováto tabulka by byla dosti paměťově náročná ($O(N^3*N^2)$) je využito toho,že uložené informace jsou binárního charakteru, tj. trojúhelník s hranou se protíná/neprotíná. Výsledná tabulka je tedy uložena v binární formě (viz obr. A2.2.8), což v praxi znamená, že hodnotu indexu sloupce je třeba převést na dvě proměnné – index slova a index bitu ve slově. Tyto dvě proměnné udávají potom jednoznačně příslušnou hranu. Jelikož navrhovaný algoritmus byl optimalizován pro počítače pracující na 32 bitové platformě, byla i velikost slova zvolena 32 bitů. Uvedené proměnné lze potom určit například následujícím způsobem:

$$i_{slovo} = index >> 5$$
$$i_{bit} = index \,\&\, \$01F \qquad \text{(A1.2.2)}$$

kde *index* je původní hodnota indexu sloupce, operace >> představuje binární operaci posunu vpravo a operace & reprezentuje operaci logického součinu.



Obr. A2.2.8 Popis struktury položek pro uzel generovaného stromu

Asi poslední věcí, kterou je třeba v této části popsat je algoritmus popisující způsob provedení expanze uzlu. V podstatě se jedná o lehce pozměněnou část algoritmu původní verze metody vkládání trojúhelníků. S využití znalosti požadované struktury uzlu lze daný postup popsat způsobem, jak je uvedeno na obrázku A2.2.9. Vstupním bodem algoritmu je uzel, pro který má být provedena expanze, a předpokládá se, že všechny položky ve struktuře popisující uzel jsou vyplněny.

1. Načtení uzlu, pro který má být provedena expanze

2. Vybrání první uvedené hrany z množiny ***setP*** daného uzlu. K této hraně se budou vkládat přiléhající trojúhelníky, uvedené v tabulce edgeTrian

3. Nastavení ukazatele do tabulky edgeTrian (dále jen *ukazatel)*. Tento ukazatel udává, který trojúhelník přiléhající k příslušné hraně má být vložen do triangulace. Jeho hodnota je nastavena na hodnotu uloženou v ***iter*** daného vstupního uzlu

4. Nastavení maximální hodnoty (dále jen *max*), kterou nemůže ukazatel již nabývat. Hodnota je nastavena podle obsahu položky ***max_iter*** vstupního uzlu

(pokračování na další stránce …)

5. Načtení uzlu, pro který má být provedena expanze

   a. Vytvoření kopii vstupního uzlu

   b. Ve vytvořené kopii se změní jednotlivé položky následovně:

      - Do pole ***T*** se přidají hrany daného trojúhelníka, které v něm ještě nejsou zapsány
      - Množina tzv. hraničních polygonů ***setP*** je modifikována na základě vložení trojúhelníku do triangulace (viz kapitola 3.5. v DP)
      - Nastavení počtu polygonů (položka ***numP***) zapsaných v poli ***setP***
      - Určení pro první uvedenou hrana v množině ***setP*** zda lze vkládat trojúhelníky zprava nebo zleva a nastavení podle toho hodnot ***iter*** a ***max_iter*** (tj. odkazy do tabulky ***edgeTrian*** na seznam přiléhajících trojúhelníky k příslušné hraně – hodnoty v podstatě udávají od kterého trojúhelníku do které lze provést další expanzi)
      - Zapsání modifikované kopie uzlu na konec tabulky úloh (viz algoritmus na obr. A2.2.3.)

6. Zvýšení ukazatel o jedničku

7. Dokud ukazatel<max, skok na bod **5**

8. Konec algoritmu

Obr. A2.2.9 Algoritmus popisující expanzi uzlu s uložením nově vzniklých uzlů na konec tzv. tabulky úloh (viz algoritmus uvedený na obr. A2.2.3)

A2.2.2. Algoritmus procházení stromu od daného uzlu

Doposud jsme se zabývali algoritmem pro vytvoření úloh na straně řídícího počítače. Nyní se již dostáváme k druhé části, a to k popisu algoritmu zpracování dané úlohy na straně počítače – „dělníka“.
V podstatě tento algoritmus byl již popsán. Jedná se totiž o totožný algoritmus s původní verzí (viz obr. 3.5.2 v DP). Jsou zde provedeny pouze dvě nepatrné změny.
První změnou spočívá pouze v tom, že veškerá potřebná předzpracování algoritmu (body 1 až 4) se provádějí již v bloku *InitSlaveDLL* (viz obr. A2.2.1.). Samozřejmě pro zajištění korektnosti informací uložených v uzlech obdržených od řídícího počítače ke zpracování, je třeba, aby byly vygenerovány všechny tabulky stejným způsobem jako na straně řídicího počítače.

Druhá změna zasahuje již do jádra samotného algoritmu. Oproti původní verzi je nutné, nastavit příslušný uzel jako výchozí bod algoritmu. V pátém bodě algoritmu není tedy do pole **setP** vložena konvexní obálka, ale je tam zkopírován obsah pole **setP** daného uzlu, a v šestém bodě je obsah celého uzlu uložen do zásobníku.

A2.3. Implementace algoritmu pro nalezení MWT

Pro otestování daného algoritmu v distribuovaném prostředí byla opět provedena implementace pro nalezení MWT (Minimum Weight Triangulation), tj. triangulace s nejmenším součtem ohodnocení jednotlivých hran. Jako ohodnocující funkce byla zvolena délka hrany v Euklidovském souřadném systému.

A2.3.1. Volba optimálního počtu předzpracovaných uzlů stromu

Strategií procházení stromu do šířky dostáváme množinu ještě nezpracovaných uzlů, které by měli být přibližně stejně výpočetně náročné (viz kapitola A2.2.1.). Nyní je ještě třeba určit, kolik je potřeba vygenerovat nezpracovaných stavů (uzlů), aby byla zaručena dobrá efektivita výpočtu. Je nutné předpokládat, že výpočet bude distribuován v síti, která je heterogenní (tj. je složena z různých typů počítačů) a jednotlivé počítače jsou propojeny komunikačními linkami s různou přenosovou rychlostí. Volba optimálního počtu předzpracovaných uzlů v takovéto síti bude závislá na následujících faktorech:

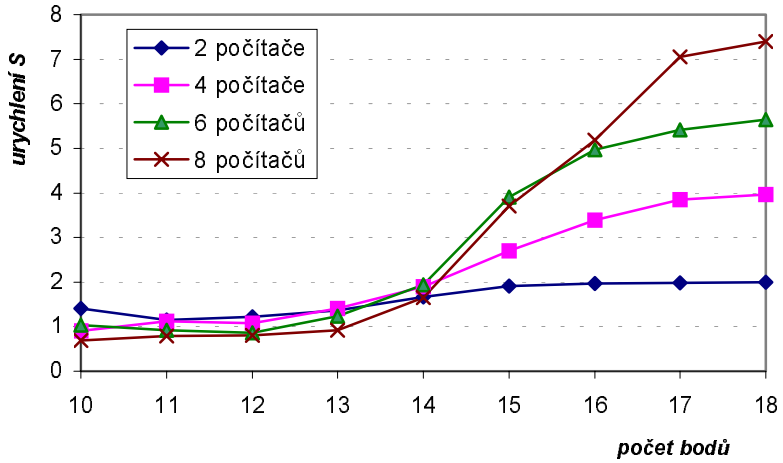
- počtu a typu použitých počítačů pro výpočet
- rychlosti komunikace mezi jednotlivými počítači
- počtu zadaných bodů, pro které má být nalezena příslušná triangulace
- topologickém rozložení bodů v rovině

Je vidět, že nastavení optimálního počtu vygenerovaných uzlů v sekvenční části algoritmu je závislý na řadě faktorů, které jsou těžko matematicky vyjádřitelné. Proto nejlepším prostředkem pro volbu vhodné hodnoty je provést několik testů v síti, na které by měl být výpočet proveden, a z výsledků odhadnout hledané optimum.
Výsledné charakteristiky závislosti na době výpočtu, které byly změřeny, jsou uvedeny díky své rozsáhlosti v příloze B. Celá příloha se v podstatě skládá ze dvou sekvencí grafů. Z první sekvence grafů (viz obr. B.1) lze vyčíst změny hledaného optima v závislosti na počtu použitých počítačů k výpočtu (testováno pro dvě různé množiny bodů). Druhá sekvence charakteristik (viz obr. B.2) se snaží zachytit závislost změny hledaného optima, jak na počtu zadaných bodů, tak i na jejich topologickém rozmístění.

Všechny testy proběhli v homogenní síti (počítače CeleronII, 566MHz, 128 MB RAM), propojenými komunikačními linkami s přenosovou rychlostí 100Mbps.
Bohužel z uvedených grafů, zachycujících závislost doby výpočtu na volbě počtu předzpracovaných uzlů, nelze stanovit přesnější závislost na jednotlivých faktorech. Hlavním důvodem je především silný vliv topologického rozmístění bodů.
Při volbě optima byl tedy proveden odhad, na základě získaných průměrných hodnot. Jeho hodnota byla odhadnuta na 800 uzlů, které je třeba vytvořit před distribucí výpočtu. Je nutné ještě poznamenat, že tato hodnota se vztahuje k dané počítačové síti. Při distribuci výpočtu v jiném prostředí tato hodnota může být odlišná.

A2.3.2. Rychlost nalezení MWT s využitím distribuce výpočtu

Jak již bylo zmíněno, pro otestování funkčnosti metody vkládání trojúhelníků v distribuovaném prostředí, byl opět celý program navržen pro hledání MWT. Pro testování bylo použito počítačů tvořící homogenní síť (tj. všechny počítače byly stejného typu - CeleronII, 566MHz, 128 MB RAM), které byly propojeny komunikačními linkami s přenosovou rychlostí 100Mbps. Počet uzlů vytvořených v předzpracování na řídícím počítači byl nastaven na konstantní hodnotu 800 uzlů (viz kapitola A2.3.1).
Pro měření, k jak velkému urychlení došlo pomocí distribuce výpočtu na více počítačů, byly neměřeny nejprve časy při distribuci úlohy na jeden počítač. Ty pak byly brány jako porovnávací vůči časům naměřených při distribuci na více počítačů. Výsledné charakteristiky urychlení jsou zobrazeny na obrázku A2.3.1, ve kterých je zachycen vliv urychlení výpočtu na počtu použitých počítačů a na velikosti zadané množiny bodů.



Obr. A2.3.1 Závislost urychlení doby výpočtu na počtu počítačů a na velikosti zadané mn. bodů

Urychlovací koeficient S je počítán podle následujícího vzorce:

$$S = \frac{t_1}{t_n} \qquad \text{(A2.3.1)}$$

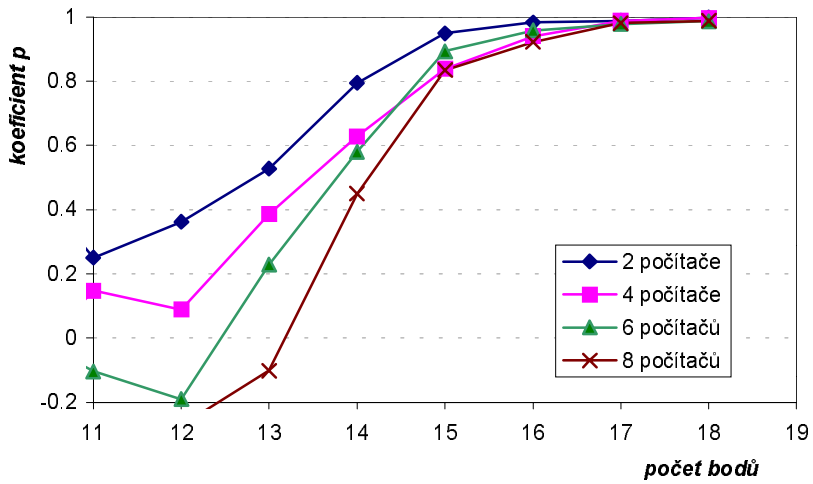
kde t_1 je čas potřebný pro výpočet při distribuci na jeden počítač a t_n je čas potřebný pro výpočet při distribuci výpočtu na n počítačích.
Maximální teoreticky možné urychlení, které je možné dosáhnout je rovno počtu použitých počítačů (pokud jsou samozřejmě všechny počítače stejného typu). Se zvyšujícím se počtem bodů vstupní množiny je z grafu patrné, že se jednotlivé křivky k těmto hodnotám přibližují, což je z našeho hlediska dobré.
U menších datových množin se vyskytuje jev, kdy distribuce na více počítačů nevedla k valnému zrychlení doby výpočtu. Dokonce došlo někde i k jeho mírnému zpomalení. To mohlo být způsobeno následující okolnostmi:

- nepřesností měření doby výpočtu (pro menší počty bodů jsou hodnoty naměřených časů velmi malé)
- časovou prodlevou vzniklou komunikací mezi počítači
- nevhodnou volbou počtu stavů vygenerovaných v sekvenční části programu

Na základě Amdahlova zákona, který byl uveden v kapitole 6.4 diplomové práce lze také určit do jaké míry se podařilo danou úlohu paralelizovat. Vyjádřením z původní rovnice proměnné $f = 1 - f$, získáme rovnici, která bude vyjadřovat z kolika procent byl daný algoritmus paralelizován na p strojích. Výsledný vztah má následující tvar:

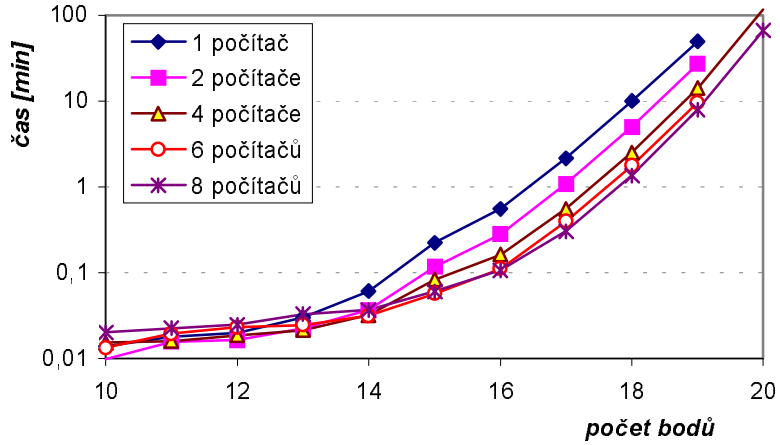
$$f' = \frac{p \cdot (1 - S)}{S \cdot (1 - p)} \qquad \text{(A2.3.2)}$$

kde S je naměřené urychlení výpočtu a p udává, kolik počítačů bylo použito k výpočtu.
Výsledný graf získaný z naměřených hodnot si lze prohlédnout na obrázku A2.3.2. Z uvedených charakteristik je vidět, že se zvyšujícím počtem bodů se hodnota míry paralelizace přibližuje k maximální hodnotě, tj. k hodnotě jedna. To svědčí o tom, že pro rozsáhlejší množiny bodů je tato úloha dobře paralelizovatelná. Pro menší množiny bodů, křivky již nevycházejí tak pěkně, dokonce se hodnoty dostávají i do záporných čísel. Tento jev je způsoben stejnými vlivy, jako bylo uvedeno při posouzení urychlení pro menší množiny bodů.

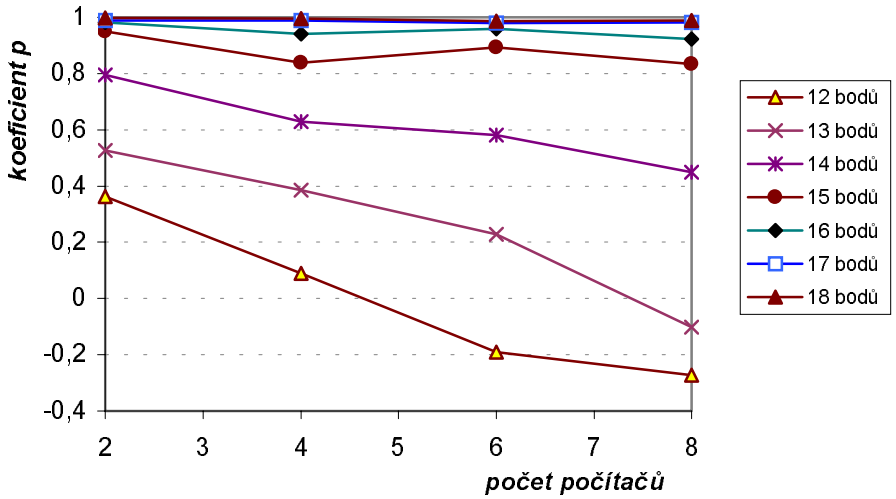


Obr. A2.3.2 Závislost míry paralelizovatelnosti algoritmu na velikosti zadané množiny bodů při různém počtu použitých počítačů k výpočtu

Při posuzování míry paralelizovatelnosti úlohy, je třeba také vzít v úvahu i čas potřebný pro výpočet. Jelikož například časová prodleva jedné sekundy se projeví jinak pro 15 bodů, kde se naměřený čas pohybuje řádově v sekundách a jinak pro 18 bodů, kde se čas potřebný pro výpočet pohybuje již v desítkách minut. Pro lepší posouzení je na obrázku A2.3.3. zobrazeny průměrné naměřené hodnoty doby trvání výpočtu, z kterých byly zbylé grafy odvozeny. Dále je pak na obrázku A2.3.4 ještě zobrazen graf obdobný grafu z obrázku A2.3.2 (míra paralelizovatelnosti výpočtu) pouze s tím rozdílem, že na x-ové ose je místo velikosti bodů vynesen počet počítačů.



Obr. A2.3.3 Závislost doby trvání výpočtu na velikosti vstupní množiny pro různé počtu použitých počítačů k výpočtu



Obr. A2.3.4 Závislost míry paralelizovatelnosti algoritmu na počtu počítačů pro různé velikosti zadané množiny bodů

Z posledního grafu (viz obr. A2.3.4) si lze povšimnout, že s přibývajícím počtem počítačů, na kterých je výpočet spuštěn klesá hodnota koeficientu *p*, který vyjadřuje do jaké míry se podařilo danou úlohu paralelizovat. Z uvedených křivek je i patrné, že rychlost tohoto poklesu je závislá na velikosti vstupní množiny bodů. Hlavní příčinou, že vůbec k danému poklesu dochází, je především nevhodná volba počtu předzpracovaných uzlů, která je zde pro všechny měření stejná (800 uzlů).

Závěrem k celé problematice týkající se distribuce metody vkládání trojúhelníků lze říci následující. Za pomoci předzpracování části generovaného stromu v sekvenční části programu, není problém danou úlohu paralelizovat. Ovšem otázka je, jak vhodně zvolit optimální počet uzlů, které mají být v této části předzpracován. Volba optimální hodnoty je závislá na řadě faktorů, které nelze jednoduše matematicky vyjádřit. Nezbývá zde tedy nic jiného než danou hodnotu na základě několika měření alespoň odhadnout. Ovšem přes to všechno se nemění nic na skutečnosti, že se jedná o NP problém. Distribucí úlohy na více počítačů, umožní pouze posunout hranici velikosti množiny bodů, pro které jsme schopni najít řešení, o něco výše.

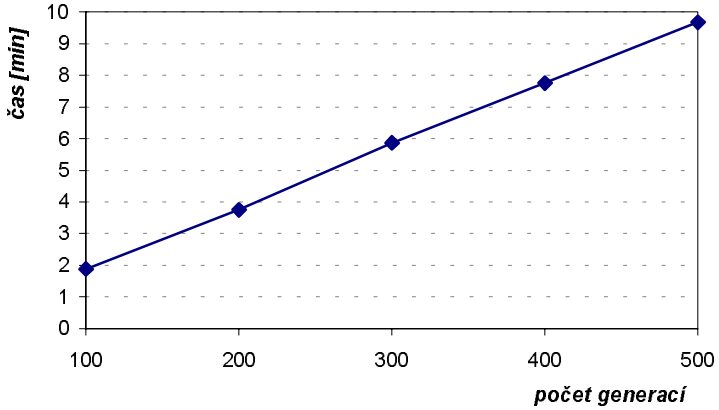
A3. Praktické využití

Jak již bylo řečeno v úvodu diplomové práce, tato úloha byla navržena z důvodu umožnění vytvoření množin triangulací, o kterých by bylo známo, že splňují dané kritérium nejlépe. Tyto triangulace by bylo pak možné použít pro posouzení heuristických metod, u kterých není zaručeno, že nimi nalezená triangulace je právě ta požadovaná.

Pro praktické ověření využití získaných výsledků, bylo vygenerováno několik datových množin a pro ně byly nalezeny patřičné triangulace s minimálním součtem délek hran (MWT). Jako heuristická metoda hledající MWT byla vybrána metoda založená na genetickém algoritmu.. Podrobný popis této metody s doporučenými hodnotami nastavení příslušných parametrů lze nalézt v [Kol01].
Při testování heuristické metody byly tedy veškeré parametry genetického algoritmu (tj. pravděpodobnost křížení, pravděpodobnost mutace, počet generací a počet triangulací generovaných v jedné generaci) nastaveny na doporučené hodnoty a poté nalezeny triangulace pro patřičné množiny bodů. Jako výchozí triangulace byla vzata tzv. žravá triangulace, o které je známo, že od hledaného MWT se vzdaluje se složitostí $O(N^{1/2})$, kde N je počet bodů.
Pro lepší posouzení dané heuristické metody bylo i odzkoušeno jakým způsobem se projeví vliv změny jak počtů generací, tak i počtu generovaných triangulacích v jedné generací (dále jen počtu triangulací).
Jako první byly tedy nalezeny triangulace pomocí genetického algoritmu s následujícím nastavením jednotlivých parametrů:

- pravděpodobnost mutace $p_m = 0{,}001$
- pravděpodobnost křížení $p_c = 0{,}25$
- počet triangulací tr = 500
- počet generací gen = 100, 200, 300, 400, 500

Časová charakteristiky doby trvání výpočtu při změně hodnoty počtu generací si lze prohlédnout na obrázku A3.1 (testováno na počítači firmy DELL, 2x450MHz, 1GB RAM) Je nutné poznamenat, že obecně časová složitost algoritmu není závislá na velikosti vstupní množiny (tj. počtu bodů), ale závisí na počtu generací a generovaných triangulací. Ovšem nastavení těchto dvou hodnot by již mělo být ovlivněno velikostí vstupní množiny, což by díky těmto testům mělo být i ověřeno.

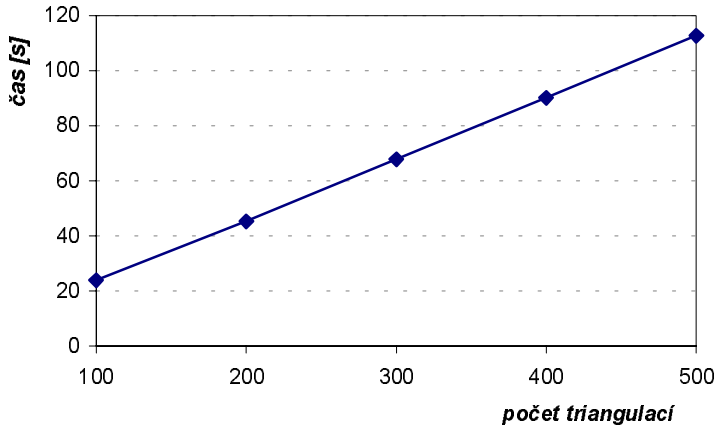


Obr. A3.1 Závislost doby výpočtu generického algoritmu na počtu generací (zbylé parametry nastaveny na hodnoty tr=500, t_m=0,001, t_c=0,25)

Bohužel při daných nastavení jednotlivých parametrů se podařilo ve všech případech vždy nalézt MWT. Ovšem z tohoto nelze vyvozovat žádné závěry, protože největší testovaná množina obsahovala pouze 23 bodů. Lze pouze říci, že pro takovéto malé množiny bodů (tj. přibližně do 25 bodů) je plně postačující provést 100 generací (při daném nastavení zbylých parametrů) a nalezená triangulace bude s vysokou pravděpodobností MWT.
Při dalším testování vlivu počtu triangulací na nalezenou triangulaci bylo zvoleno následující nastavení jednotlivých parametrů:

- pravděpodobnost mutace $p_m = 0{,}001$
- pravděpodobnost křížení $p_c = 0{,}25$
- počet triangulací tr = 100, 200, 300, 400, 500
- počet generací gen = 100

Vliv změny počtu triangulací na dobu výpočtu je zobrazen na obrázku A3.2 (testováno na stejném počítači).



Obr. A3.2 Závislost doby výpočtu generického algoritmu na počtu triangulací (zbylé parametry nastaveny na hodnoty gen=100, t_m=0,001, t_c=0,25)

Daná nastavení počtu triangulací se již nepatrně projevila na nalezených triangulacích. Pro posouzení přesnosti nalezené triangulace vůči MWT byl definován poměrový koeficient ν, který je počítán podle následujícího vzorce:

$$\nu = \frac{weight(T_{GA})}{weight(MWT)} \qquad \text{(A3.1)}$$

kde funkce *weight* je ohodnocující funkce triangulace (v tomto případě se jedná o součet délek hran triangulace), T_{GA} je triangulace nalezená genetickým algoritmem a *MWT* je triangulace s minimálním součtem délek hran.

V následující tabulce (viz obr. A3.3) jsou uvedeny průměrné hodnoty koeficientu ν pro daná nastavení počtu triangulací.

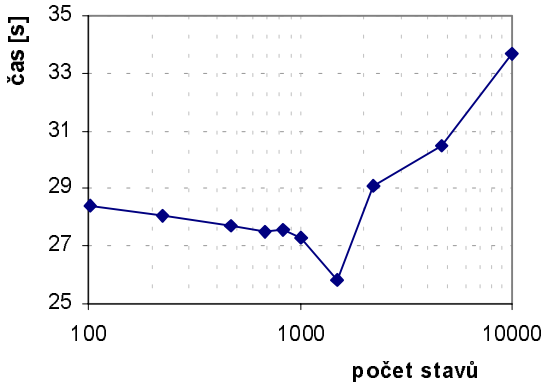
tr	počet bodů								
	15	16	17	18	19	20	21	22	23
100	1	1	1	1	1	1,00003	1	1,0004	1
200	1	1	1	1	1	1,00003	1	1,0004	1
300	1	1	1	1	1	1	1	1	1
400	1	1	1	1	1	1	1	1	1
500	1	1	1	1	1	1	1	1	1

Obr. A3.3 Tabulka s hodnotami koeficientu ν závislosti na počtu triangualcí a na velikosti množiny bodů (pro gen=100, t_m=0,001, t_c=0,25)

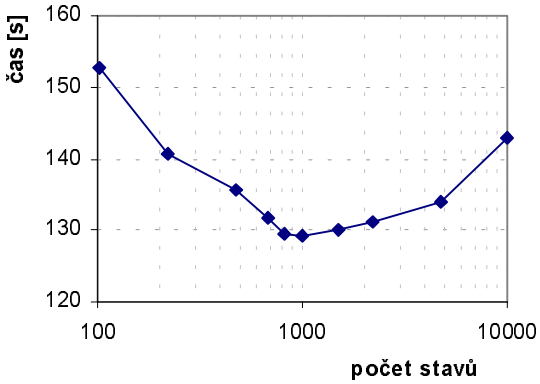
Bohužel z uvedených hodnot nelze mnoho vyčíst. Pouze je zde již nepatrně naznačena závislost nastavení parametru počtu triangulací na počtu bodů. Ovšem více toho nelze říci.
Závěrem lze konstatovat, že velikosti množin bodů, pro které jsme schopni nalézt MWT brutální silou, jsou dosti malé a zatím pro praktické využití nepostačující.

Příloha B – Charakteristiky

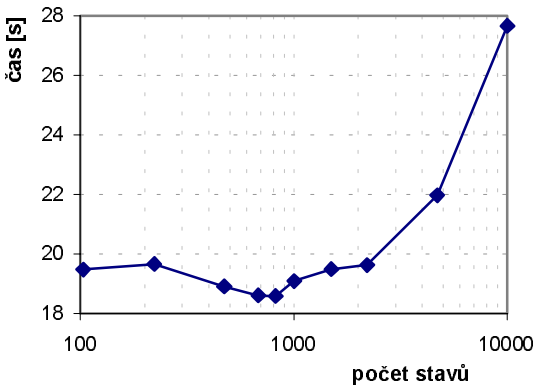
V této příloze jsou uvedeny charakteristiky, popisující časovou závislost doby výpočtu potřebné pro nalezení MWT (v distribuovaném prostředí) pomocí metody vkládání trojúhelníků v závislosti na počtu předzpracovaných uzlů generovaného stromu (viz kapitola A2.3.1.). Tyto charakteristiky jsou vhodné pro odhadnutí optimální hodnoty počtu uzlů, které je třeba předzpracovat, než bude výpočet distribuován.
Všechny uvedené charakteristiky byly změřeny na homogenní počítačové síti. Výpočet proběhl na počítačích typu CelleronII, 566MHz, 128MB RAM. Pro komunikace mezi jednotlivými počítači byla použita komunikační linka s přenosovou rychlostí 100Mbps.



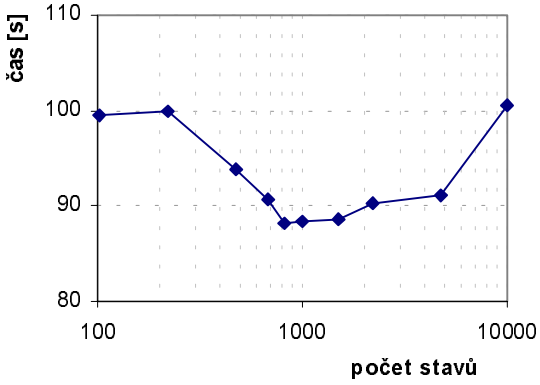
a) 4 počítače, 16 bodů



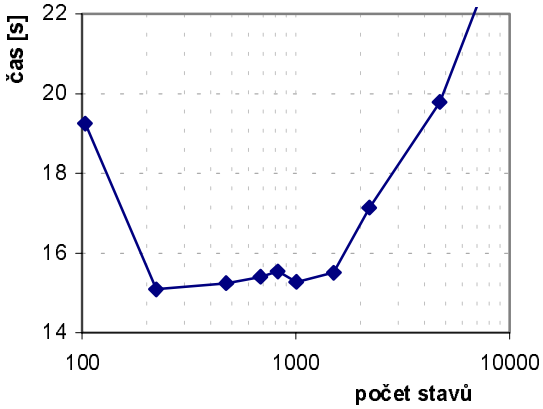
b) 4 počítače, 17 bodů



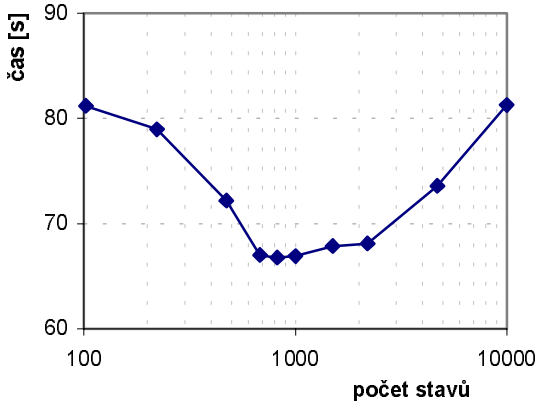
c) 6 počítačů, 16 bodů



d) 6 počítačů, 17 bodů

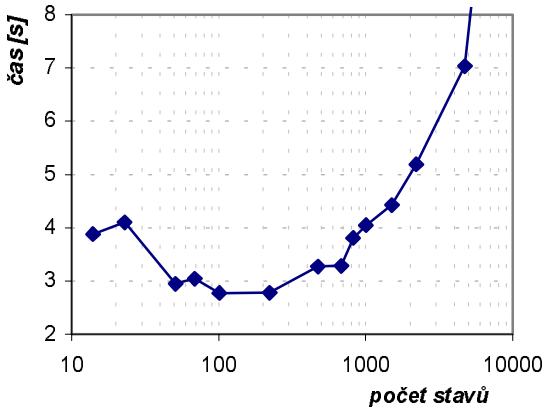


e) 8 počítačů, 16 bodů



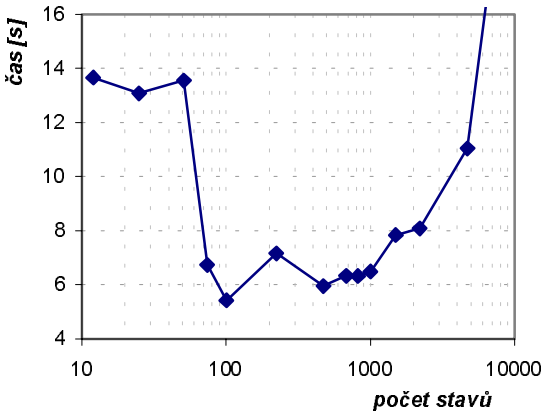
f) 8 počítačů, 17 bodů

Obr. B.1 Závislost doby výpočtu na počtu předzpracovaných stavů při různém počtu použitých počítačů k distribuci výpočtu. Grafy nalevo jsou pro množinu 16 bodů a grafy napravo jsou pro množinu 17 bodů.

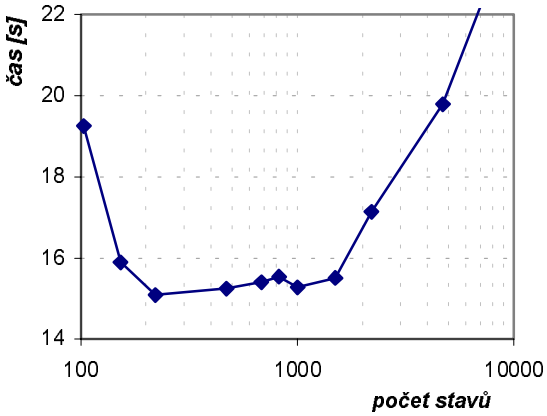


a) první množina - 15 bodů

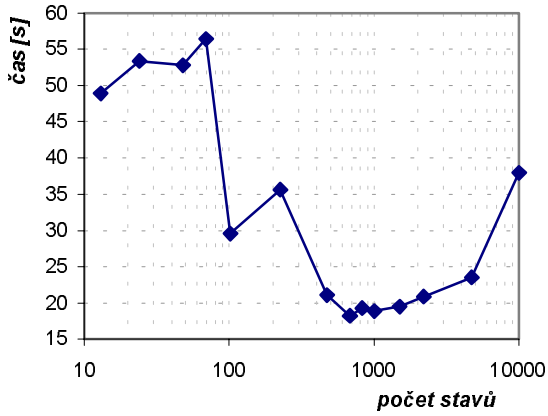
b) druhá množina - 15 bodů



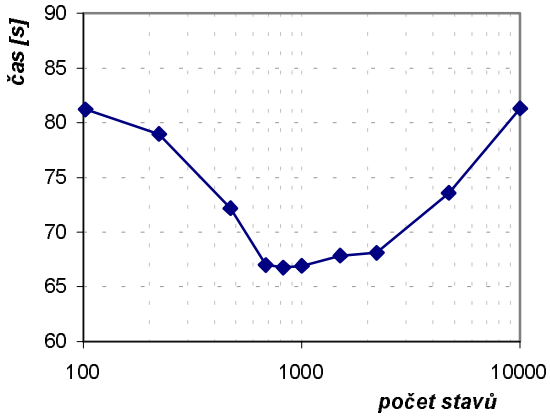
c) první množina - 16 bodů



d) druhá množina - 16 bodů



e) první množina - 17 bodů



f) druhá množina - 17 bodů

Obr. B .2 Závislost doby výpočtu na počtu předzpracovaných stavů při různé velikosti vstupní množiny bodů. Distribuováno na 8 počítačích.

Příloha C – Odhad doby výpočtu

V diplomové práci na straně 42 (viz [Hla01]) byl uveden graf, na kterém byly zobrazeny časové průběhy doby výpočtu pro jednotlivé metody. V této příloze jsou k těmto časovým průběhů uvedeny funkce, které aproximují dané hodnoty. Na základě těchto funkcí lze pak učinit přibližný časový odhad pro rozsáhlejší množiny bodů, které v grafu nejsou uvedeny.
Aproximující funkce je hledána ve tvaru:

$$f_t(N) = a \cdot m^N \qquad \text{(C1.1)}$$

kde a a m jsou konstanty charakterizující průběh dané křivky a N je počet bodů. Konkrétní rovnice pro jednotlivé metody, jsou následující:

- metoda odebírání hran z úplného grafu

$$f_t(N) = 1{,}36 \cdot 10^{-16} \cdot 11{,}621^N \quad [\text{min}]$$

- metoda vkládání hran do prázdného grafu

$$f_t(N) = 1{,}38 \cdot 10^{-14} \cdot (12{,}899)^N \quad [\text{min}]$$

- metoda generování kombinací I

$$f_t(N) = 8.11 \cdot 10^{-19} \cdot (84{,}225)^N \quad [\text{min}]$$

- metoda generování kombinací II

$$f_t(N) = 3{,}12 \cdot 10^{-16} \cdot (15{,}189)^N \quad [\text{min}]$$

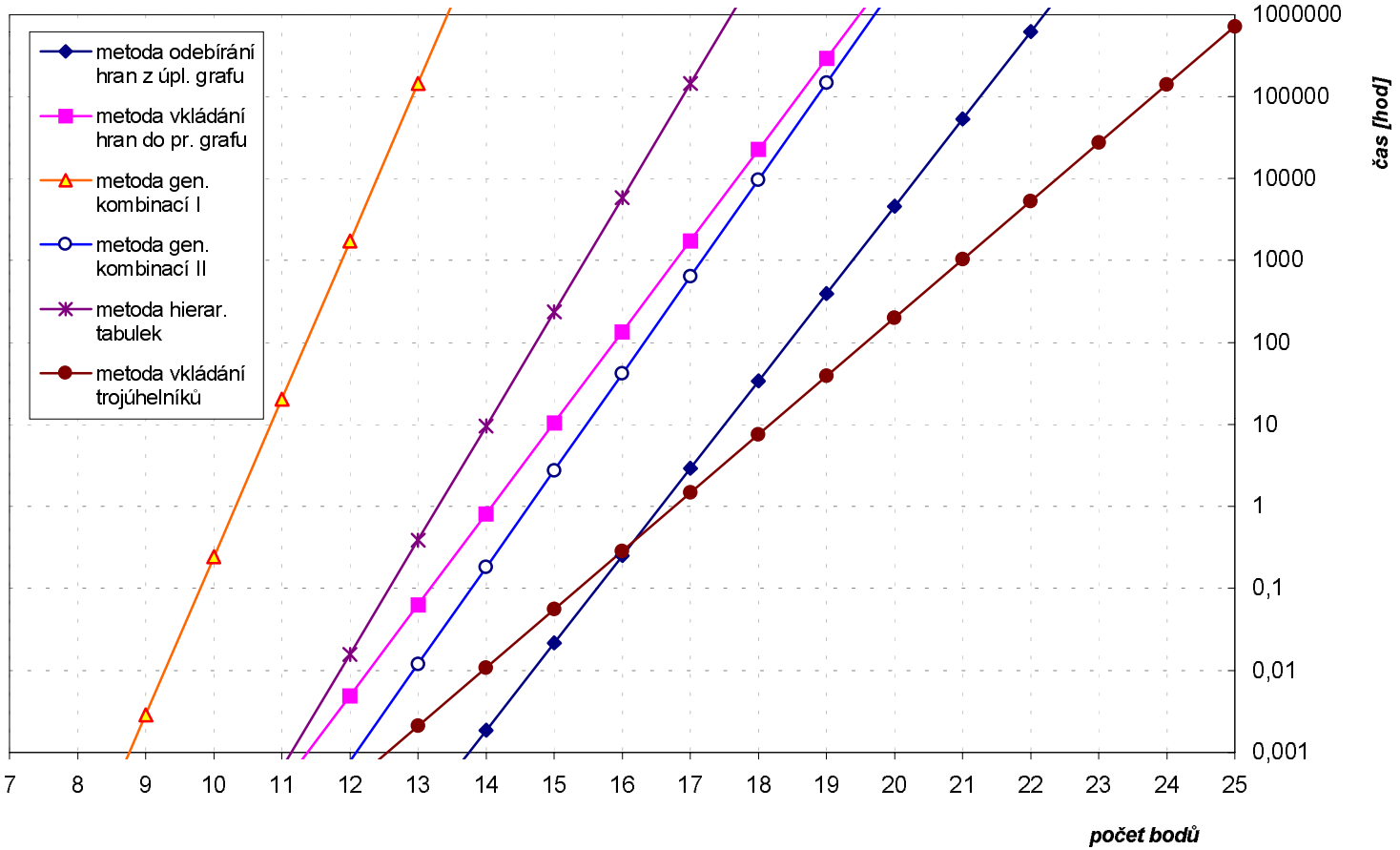
- metoda hierarchických tabulek

$$f_t(N) = 1{,}797 \cdot 10^{-17} \cdot (24{,}725)^N \quad [\text{min}]$$

- metoda vkládání trojúhelníků

$$f_t(N) = 7{,}08 \cdot 10^{-11} \cdot (5{,}145)^N \quad [\text{min}]$$

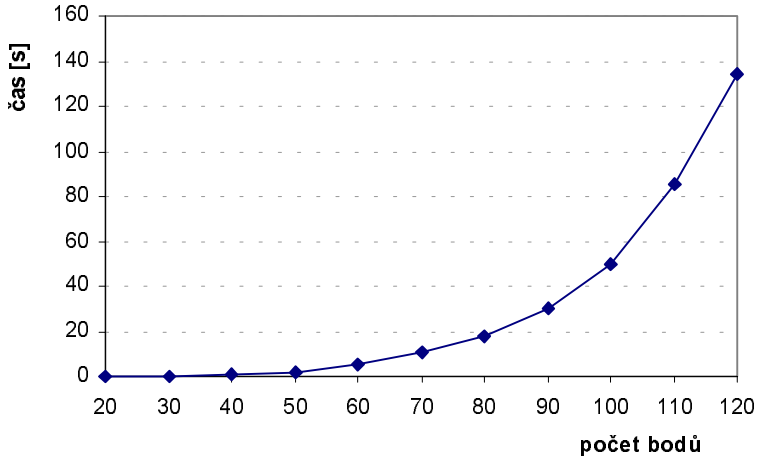
Průběhy uvedených funkcí si lze prohlédnout v grafické podobě na následujícím obrázku (osa y byla pro lepší přehlednost převedena z minut do hodin).



Obr. C.1. Průběhy funkcí aproximující naměřené hodnoty doby potřebné pro výpočet u jednotlivých metod v závislosti na počtu bodů

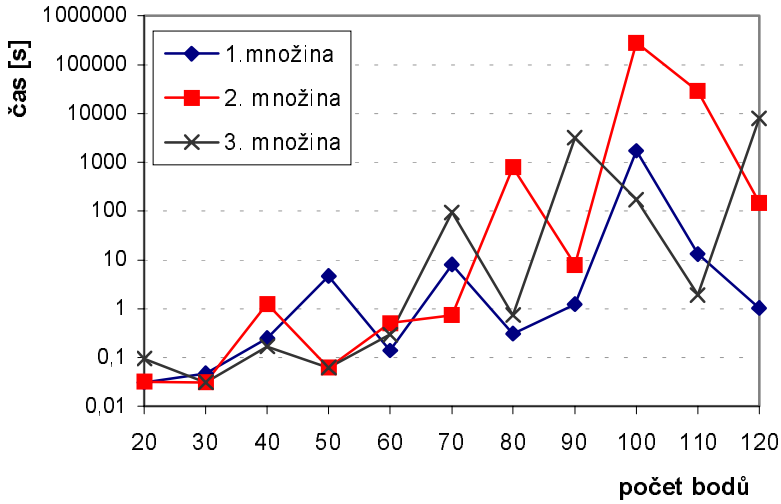
Příloha D – LMT skeleton

Podrobný popis LMT skeletonu lze najít v [Dic96], [Bei97] a řadě dalších publikací. Jeho význam je především v oblasti konstrukce MWT (*Minimum Weight Triangulation*), kde bylo vypozorováno, že jeho hrany jsou vždy součástí MWT (dosud nebyla nalezena žádná množina bodů, pro kterou by toto neplatilo).
Díky tomu, že se LMT skeleton jeví jako velmi dobrý (nalezne velkou část MWT), byl otestován na modifikované *Metodě vkládání trojúhelníků* používající hash tabulku. Pro dosažení co nejlepších výsledků, byly nastaveny parametry algoritmu na základě předchozích experimentů na hodnoty SIZE = 2^{16} a MAX = 10^6 (viz 6.kapitola). Testy proběhly na stejném počítači jako v předchozích případech - DELL, PentiumIII, 2x450MHz, 1GB RAM, OS Windows 2000.
S použitím LMT skeletonu v předzpracování se již podařilo při prvních testech najít MWT pro značně větší množiny bodů než bez něj. Doba předzpracování, která byla dříve zanedbatelná, sice nyní roste v závislosti na počtu bodů mocninou řadou (viz obrázek D.1), ale díky tomu, že složitost algoritmu má exponenciální charakter je tato skutečnost nepodstatná.



Obr. D.1 Průměrný čas potřebný pro nalezení LMT skeletonu v závislosti na počtu bodů

Při hodnocení výsledků metody vkládání trojúhelníků bylo již zmíněno, že čas potřebný pro průchod algoritmem je závislý, jak na počtu bodů, tak i na jejich topologickém rozložení. Nyní k tomuto ještě přibývá faktor udávající počet nalezených hran pomocí LMT skeletonu. Čím více se podaří skeletonem najít hran (opět záleží na topologickém roložení bodů), tím menší bude vstupní množina hran, resp. trojúhelníku algoritmu a tím rychlejší bude i výpočet.
O správnosti tohoto tvrzení se lze přesvědčit na obrázku D.2, kde jsou uvedeny charakteristiky doby průchudu algoritmem pro tři různé náhodně vygenerované množiny bodů. Závislost na počtu bodů je konstruována stejným principem jako v předchozích případech, tj. do počáteční množiny jsou postupně přidávány náhodně vygenerované body.



Obr. D.2 Příklady doby potřebné pro průchod algoritmem Metody vkládání trojúhelníků s pamatováním si lok. optim v závislosti na počtu bodů pro různé množiny (SIZE = 65536, MAX = 1000000)

Z výsledných charakteristik je patrné, jak u všech testovaných množiny doba potřebná pro výpočet značně kolisá Toto je způsobeno především topologickým rozložení bodů, které se promítne, jak v úspěšnosti LMT skeletonu, tak i v rychlosti samotného algoritmu. Přesto lze říci, že uvedené průběhy by se dali aproximovat exponenciálními křivkami.

Tedy je zde opět patrná nepolynomiální složitost úlohy. Ovšem nyní se již pohybujeme na hranici 100-150 bodů (z původních 25-30), což by se dalo nazvat značným pokrokem. Jediným úskalým je puze to, že LMT skeleton lze použít pouze pro MWT a neplatí obecně pro libovolné kritérium.