# Using Layout Stitching to create deterministic Local Graph Layouts

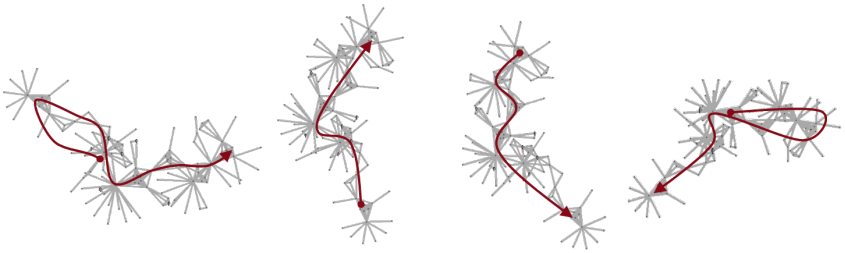
Martin Steiger
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt
Germany
martin.steiger@
igd.fraunhofer.de

Hendrik Lücke-Tieke
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt
Germany
hatieke@
igd.fraunhofer.de

Thorsten May
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt
Germany
thorsten.may@
igd.fraunhofer.de

Arjan Kuijper
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt
Germany
arjan.kuijper@
igd.fraunhofer.de

Jörn Kohlhammer
Fraunhofer IGD
Fraunhoferstr. 5
64283 Darmstadt
Germany
joern.kohlhammer@
igd.fraunhofer.de



## ABSTRACT

Dynamic graph layouts are often used to position nodes in local views of large graphs. These layouts can be optimized to minimize changes when navigating to other parts of the graph. Dynamic graph layout techniques do not, however, guarantee that a local layout is recognizable when the user visits the same area twice. In this paper we present a method to create stable and deterministic layouts of dynamic views of large graphs. It is based on a well-known panorama-stitching algorithm from the image processing domain. Given a set of overlapping photographs it creates a larger panorama that combines the original images. In analogy to that our algorithm stitches pre-computed layouts of subgraphs to form a larger, single layout. This deterministic approach makes structures and node locations persistent which creates identical visual representations of the graph. This enables the user to recognize previously encountered parts and to decide whether a certain part of a dataset has already been explored before or not.

## Keywords

dynamic graph, explorative analysis, mental map, graph layout stitching

## 1 INTRODUCTION

Showing the structure emerging from the network connections is one goal of graph visualization. However, human's visual intelligence can be used only if adequate data displays are provided. Using node-link diagrams is a popular visualization technique that works particularly well for small to medium-sized graphs. The goal of our technique is to ease the exploration of local structures of large static graphs based on a node-link diagram.

A typical user task is the exploration of the neighborhood around a focal area of the graph. We consider such an exploration as success if the user is able to mentally chart the visible parts of the graph. Thus, it increases the area the user is familiar with. To be precise, "familiarity" reflects two abilities to us: Firstly, the user is able to recall a visible area upon revisiting. Secondly, the user is able to mentally extend this area beyond the visible part enabling the user to plan and predict navigation. Revisitation has been identified by Lee et al. [LPP+06] as one of the tasks to be supported by graph visualization.

The tasks we support with our technique are characterized by the following two assumptions: The first one is



 ISBN 978-80-86943-74-9

that the information on the local level is more important than the global structure of the graph. The second assumption is that movement along the edges is required to gather required information. An example for such tasks are investigations in citation networks or social networks. We derive two conflicting requirements from

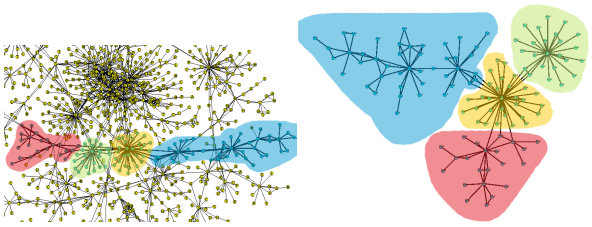


Figure 1: Identical network parts have been highlighted in a global layout (left) and a local, independent layout (right). The former display makes the impression that the clusters were linearly connected. In particular, it seems as if the shortest path from red to blue leaded through green and yellow. However, the local view reveals that this is not the case.

these goals and task characteristics. In the following, we will describe these requirements, show why they are conflicting and propose a solution to resolve this conflict as the core contribution of this paper:

The first requirement is to show only the part of the graph the user is interested in and to adapt the layout to this visible subgraph. To give an example, Figure 1 shows a global layout with five clusters in a linear arrangement. Instead, a local layout of the clusters exposes the actual topology of these clusters. This requirement is dealt with in a number of existing *dynamic layout* approaches. Dynamic layout covers techniques for the selection of interesting or important areas of the graph, for the definition of incremental layouts and the smooth transition between consecutive layouts of these visible subgraphs during exploration. We will call these visible subsets of the graph *frames* from now on. The rationale for smooth transition is to minimize the users' effort to keep track of the evolving layout. However, this applies to consecutive frames only. Revisiting a known area of the graph after extensive exploration usually results in very different layouts.

This fact leads to our second requirement. Whenever a user revisits an area of the graph for a second time the difference between the two layouts should be as small as possible. We state that this requirement applies regardless of the length or direction of the user's exploration path between any two visits of the same area. For static graphs we argue that the visible layout is determined by the currently visible subgraph only. A simple solution exists if only the second requirement were to be considered: Compute a *static* layout of the complete graph and toggle the visibility of nodes and edges as needed. However, a static layout conflicts with the first requirement, because it naturally does not adapt to local features.

Our solution is a resolution of this conflict fulfilling both requirements. We propose a dynamic layout that adapts to the currently visible subgraph, but which is independent of the exploration path. The main challenge is to keep the layouts "stable" during exploration. We use a two-level-layout strategy to solve this problem. The first-level layout is done before interactive exploration: We compute a set of overlapping subgraphs which covers the entire graph. The overlapping cover guarantees that most of the nodes will appear in at least two subgraphs. For every subgraph a layout is computed independently to produce the first-level layout. These layouts serve as "building-blocks" for the second level layout that is used during exploration.

The challenge of the second-level layout is to combine these layouts depending on a given visible area of the graph. The node coordinates from different subgraph layouts need to be merged in a deterministic fashion. To achieve this goal we choose a technique that originally comes from the field of image processing: *Panorama stitching* is used to merge a set of overlapping photographs into a single, seamless image. We transfer this technique to graphs. Depending only on the currently visible frame, individual subgraph layouts are selected, weighted and merged to produce the final, visible layout.

To the best of our knowledge, these conflicting requirements have not been solved with a single technique before. Our contribution is a technical solution serving as a proof-of-concept which fulfills both requirements. It extends the notion of layout stability from "frame-to-frame-coherence" to "frame consistency". This means that the layout of any subgraph looks the same or at least similar for every visit. As a concept, it makes use of existing approaches to create subgraphs and their layouts, but it is not bound to specific approaches. In fact, we believe that this concept offers a design space, which is worth to be explored further in future.

The rest of the paper is organized as follows: Section 2 discusses related work in the area, before the concept of our method is described in Section 3. In Section 4 we present test results for artificial and real datasets before we conclude with discussion and future work in Section 5.

## 2 RELATED WORK

In this section we first describe fundamental work on the preservation of the mental map for the navigation in network visualization, especially with respect to design considerations and criteria. After that, we present techniques which have been developed to tackle this problem by improving layouts and/or interaction.

 ISBN 978-80-86943-74-9

The preservation of the mental map of graph visualization has become an important goal ever since its introduction by Eades et al. [ELMS91]. In the literature, dynamic layout techniques have been proposed to solve two different problems: The first problem is the layout computation of dynamic graphs, which has been formalized by North [Nor96]. The second problem is the layout of a dynamic *view* of a graph, which has been described by Huang at al. [HEW98]. A dynamic view is basically a visible subgraph, which can be "moved" interactively for browsing. This problem has been applied to static graphs, for example, by Huang et al. [HEL05] and van Ham and Perer [vHP09]. According to North layout stability is achieved by minimizing layout changes between consecutive frames. Interestingly, the two problems are similar from the perspective of this criterion alone. In fact, many existing techniques could be used to solve both problems. Our definition for layout stability, however, does not apply to consecutive frames alone. In addition we require that the layout of any given subgraph will be the same upon revisitation. Hence we can only claim to solve the second problem here; the dynamic view of a static graph.

We consider the layout stability as a means to augment recall on recently visited regions of the graph. The results of Marriott et al. [MPWG12] suggest that layout features have different cognitive impact, e.g. favoring symmetry or orthogonality. In an earlier study, Purchase and Samra [SP08] note that minimal node movement may not be the most relevant criterion for mental map preservation. Archambault et al. [APP11] compare animation approaches to small-multiple approaches, but their effect on mental map preservation are inconclusive. We have to note that especially recall experiments are naturally limited to small graphs - and schemes to transfer results to real world graphs have yet to be devised.

Many dynamic layout techniques are modifications of static layout techniques which impose specific constraints or quality objectives on the transition between two consecutive layouts. Brandes and Mader [BM12] compare different measures, especially with respect to the trade-off between individual layout quality and stability between frames. They note that even slightly lowered requirements in quality often offer a significant boost in stability. Virtually all elements of a graph visualization have been covered by previous approaches to stabilize the mental model upon dynamic changes. For example, Frishman and Tal [FT08] and Erten et al. [EHK+04] propose approaches where quality objectives apply to the node movement. Frishman and Tals approach fine-tunes the inertia of nodes between consecutive frames. Erten et al. propose a natural extension to force-feedback techniques by using a (2+1)-dimension layout using virtual edges connecting different time-frames. Other approaches, like that of Dwyer et al. [DMS+08] and Frishman and Tal's [FT04] focus on the preservation of node clusters in the dynamic layout. Additionally, Dwyer's approach optimized the arrangement of polyline-edges. Aside from spring-embedding layouts, dynamic layout methods have also been used in conjunction with other techniques. For example, Görg et al. [GBPD05] use Sugiyama-style layout techniques.

Among these approaches, our technique relates most to cluster-preserving dynamic layouts. However, the "clusters" in our approach are subgraphs, which are layouted independently in a preprocessing step and merged together depending on the current area of interest of the user. Archambault et al. [AMA07] present a similar strategy with the static multi-level technique *Topolayout*. Topolayout creates hierarchical partition layouts with the most suited technique and merges them to minimize edge lengths and crossings. In contrast, our technique creates overlapping subgraphs which are dynamically merged along the overlapping nodes.

Aside from techniques which aim to preserve the mental map on a purely structural level, the role of interaction and navigation cues must be considered as well. In fact, Marriott et al. note in their study [MPWG12] that node labels are more powerful cues for mental mapping. However, we think that layout stability supports the effective use of local navigation cues like labels, because they need to be located in the view to be useful. Moscovich et al. [MCH+09], van Ham and Perer [vHP12] and May et al. [MSDK12] propose techniques to ease navigation across larger distances. Their common idea is to provide visual cues pointing to otherwise invisible nodes or regions of the graph.

# 3 CONCEPT

In this section we will describe how we derive a deterministic global layout from a set of local layouts. We therefore transfer the panorama stitching algorithm to the graph layout domain. Before we can perform our layout stitching algorithm, a set of subgraphs with overlapping node sets needs to be created. A local layout is then computed for every subgraph - independent from the rest of the graph. We refer to these subgraph layouts as *patches* from now on. We then align these patches to match the positions of all nodes that exist in more than one subgraph as good as possible. The position of nodes that exist in multiple patches are then merged and a unified layout is created. The basic idea is illustrated in Figure 2. In the final step, we will explain how to create a layout that consists of more than just two patches.

## 3.1 Definitions

We define a graph $G(V,E)$ comprising a set of vertices $V$ together with a set of edges $E$. Our method works

 ISBN 978-80-86943-74-9

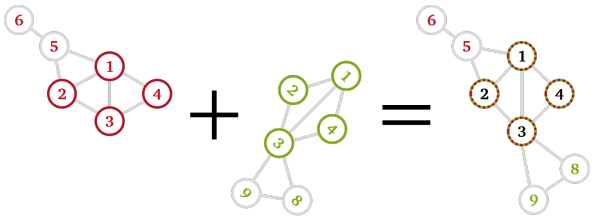


Figure 2: The green layout patch is aligned to the red patch using the four shared nodes. Nodes that exist in both layouts are merged, creating a unified layout of both patches.

with both directed and undirected edges without limitations. However, we will assume for the sake of simplicity that the graph is connected, i.e. a path exists between every pair of nodes in $G$. We also define a clustering $C(V)$ as a mapping of $V$ to a set of classes, so that every vertex in $G$ is linked to one or more classes.

## 3.2 Pre-processing

The series of visible frames is defined by the user who is browsing this graph on a local level. We do not define the means of interaction here, but we assume that the set of visible nodes can be derived from the user's interaction. Our concept defines a local layout for this subgraph. Given a set of visible nodes, the set of layout patches that need to be merged can be derived. The frame which was displayed during the last timestep does not influence the layout of the current frame. This ensures that the same picture is created – independent from the exploration path.

If no set of patches for a graph is provided, we compute a cover of overlapping subgraphs from a topology-based clustering, so that every vertex of the graph is contained in at least one patch. Our approach works with basically any clustering algorithm. However, we point out that the cluster size and content have an influence on their layout which in turn influences the cluster shape that is used for the stitching.

The resulting clusters cannot be used directly as patches, because the clustering typically creates a partition of the graph, i.e. every node is contained in exactly one single cluster only. A straight-forward approach to make them overlap is to include neighbors of the first degree. In other words: nodes from other patches that are directly connected are added to the node set of the patch. Larger sets are created by adding neighbors of neighbors and so on.

We consider two clusters to be connected if they share at least three nodes. This is the minimum number of points that is required for the layout patch alignment computation.

As soon as the subgraphs are created, a layout is computed for each of them. This can be performed completely independently which allows for using different layout algorithms. Moreover, the computation can be done in a pre-processing step, but also deferred until the layout is actually needed which avoids unnecessary computational overhead. Force-directed algorithms such as that of Fruchterman and Reingold appear to be a sound choice as they reveal local structure and are flexible to integrate user-specified requirements[Kob12].

## 3.3 Shape Matching

The sub-layouts are computed independently, therefore the position of nodes is given in a local coordinate system. Nodes that exist in more than one layout generally have different positions in each of them. We will now describe how two patches with overlapping node sets can be aligned so that the distance in between is minimized. Individual node positions do not fit perfectly, but this will be fixed in a later step.

The idea of stitching shapes is based on the work of Brown and Lowe[BL07] who describe an approach for automatic panorama stitching. The authors compute a matching transformation for images based on distinct, but overlapping point clouds. This process is far less complex for graph layouts as no image post-processing such as brightness compensation is required. Most importantly, the point correspondences in the two point sets is known in our setting which simplifies the algorithm.

The second, important contribution comes from the the shape-matching algorithm of Müller et al. which works with identical point clouds but in a very different context[MHTG05]. The authors present a method that allows for elastic deformation of three-dimensional objects. With the help of shape matching, the points of the deformed object can be gradually transformed back to their original position. For that, the two geometric point sets are compared and a transformation that reduces the pair-wise distance between all points to a minimum is deduced. Apart from translation and scaling, their transformation scheme offers refinements such as twisting and compression which are not present in the work of Brown and Lowe.

We trivially acquire a set of vertices that exist in two given layouts by computing the intersection of the two sets.

As long as the set contains at least three vertices, we can use the standard least-squares fitting method [AHB87] to compute a deterministic matching transformation. For the sake of simplicity, we will restrict this computation to rigid transformation, i.e. rotation and translation, but general affine transformations are feasible as well. For every point $p$ in sublayout $A$ we specify its counterpart $p'$ in sublayout $B$ as

$$p'_i = Rp_i + t + \varepsilon_i$$

 ISBN 978-80-86943-74-9

Figure 3: Rotating one of the two point clouds (green) reduces the average distance between pairs.

Here, $R$ is a rotation matrix, $t$ a translation vector and $\varepsilon$ the measure of error. After solving for $\varepsilon$ and accumulating the error over all points, we get

$$\varepsilon^2 = \sum^{n} \|\varepsilon_i\|^2 = \sum^{n} \|p'_i - (Rp_i + t)\|^2$$

The error becomes minimal if both point clouds have the same centroid [AHB87]. This can be achieved by subtracting the centroid of their respective sets (denoted as $c_p$ and $c_{p'}$) from the point locations. The task is now to find an optimal rotation matrix where the pairwise distance is minimal for all points. This matrix can be deduced from a $2 \times 2$ cost matrix $H$ that measures the distance between two point clouds. We subtract the centroids from both datasets to bring them to the origin and define this matrix $H$ as

$$H = \sum_{i}^{n} (p_i - c_p)(p'_i - c_{p'})^T$$

Using singular value decomposition (SVD), the matrix $H$ can be factorized into two rotations $U$ and $V$ and a diagonal scaling matrix $S$.

$$[U,S,V] = SVD(H)$$

See, for example, the introduction by Wall el al. [WRR03] for details on the mathematical background of the singular value decomposition. The final desired rotation matrix can be computed as:

$$R = VU^T$$

The final result is a transformed point $\hat{p}_i$ that represents the point $p_i$ of sublayout $A$ in the coordinate system of sublayout $B$.

$$\hat{p}_i = R(p_i - c_p) + c_{p'}$$

The accumulated difference between $\hat{p}_i$ and the original point $p'_i$ relates to the previously computed error $\varepsilon$ and can be used as a quality measure for this transformation process.

## 3.4 Combining multiple shape matching transformations

The approach we presented so far works well for combining two patches. However, in general, a frame consists of more than that. We therefore describe how to stitch multiple patches in one frame and how we create a smooth transition between two consecutive frames. The problem we solve here is to find a deterministic order in which the patches are stitched. Therefore, we create a meta-graph of the patches. Two patches are connected, if two patches share common nodes (see Figure 5 left and center). Thus, they can be stitched together. For the remaining part of the paper, this graph is referred to as *patch graph*.

If the patches that should be merged are connected directly, only a deterministic order of stitching operations needs to be defined. Otherwise, also a connecting series of patch stitchings must be generated to ensure that also distant patches can be combined.

This series of stitchings of overlapping patches can been seen as a path in the patch graph. Also, on this level of abstraction, the interactive exploration can be seen as a user-driven traversal of this patch graph.

Starting with a single patch, the user continues exploring, eventually reaching a part of the graph, that can not be visualized without including additional patches in the visible subgraph. Adjacent patches are then added until the requested graph region can be visualized. This graph traversal must be stateless and therefore independent of previously visible patches. If this was not the case, different exploration paths would have different stitching orders thus result in different global layouts. Three possible setups are depicted in Figure 4.

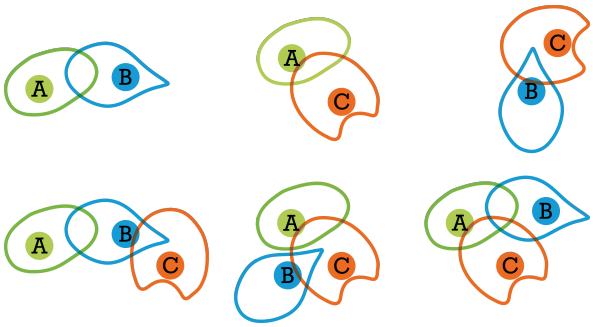



Figure 4: Three patches (A, B, C) with corresponding 1-to-1 stichtings (top row). If the patches would be stitched in the order they become visible, different stitched layouts would result. In this configuration three different global layouts could be produced (bottom row).

The reason for this is that the patch graph contains multiple paths that connect the visible patches. Reducing the number of edges naturally leads to a reduction of the number of paths. To enforce a stable matching order, we remove all edges from the patch graph that are not strictly necessary to keep the graph connected (see the illustrations in Figure 5). What is left is a spanning tree of the graph and can be computed by Kruskal's algorithm.

 ISBN 978-80-86943-74-9

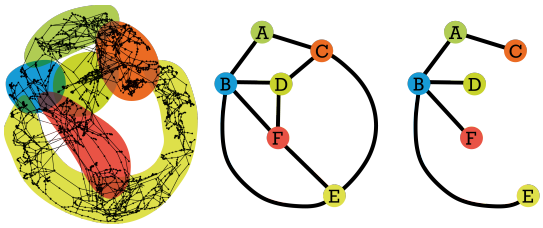



Figure 5: The original graph is reduced to a graph of patches which is then reduced to a spanning tree. This tree is used to define unique paths between any two nodes.

It is also able to incorporate edge weights, thus computing the spanning tree with lowest total weight – the minimum spanning tree (MST). Starting with a graph that has contains all nodes but no edges, edges with the lowest weight are continuously added as long as they don't lead to cycles in the graph.

Although the error value of each matching seems like a natural choice to maximize the quality of the whole layout, several drawbacks lead us to the decision against using it. First and foremost, using the matching error as edge weight is possible only if the matching error was known for all pairs of connected patches. Computing the optimal affine transformation of all possible combinations of layout patches is rather time-consuming. Furthermore, interactive manipulation of a single patch layout would result in changing weights for its incident edges, which in turn could cause changes in the spanning tree of the patch graph.

Instead, we define a similarity-based weight function so that edges between pairs of patches with large overlap ratios have lower edges weights. They are then most likely to be stitched first. The Jaccard similarity coefficient is a measure that indicates how similar two sets are and is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

This measure does not require the computation of the matching error between all edges in the patch graph. Similar to the distance function that is often used in graph clustering, we can use this (or another) similarity measure and assign this value to the edges of the patch graph.

For every pair of layout patches in this spanning tree, only one single path exists. These paths in the tree are no longer the shortest in general when compared tot the original patch graph, but this reduction in freedom results in consistent patch stitching chains. This also ensures independence of previous frames, as the stitching order is fixed. We use the root of the spanning tree as end point for all paths. Thus, every visible patch is stitched to its parent patch until the root node in encountered. This is a critical aspect as it ensures that patches are always matched to the same neighbor patches. The local position of a node is thus transformed by the series of affine transformations of the patches along the path to the root patch.

Some nodes belong to multiple patches and would, without additional correction, have multiple positions on the drawing canvas. We therefore derive from all these positions a commonly shared, unique position. In such cases, we use a linear combination of the nodes' weight factor to place the nodes depending on time and the user focus. This ensures a smooth transition from one layout frame to another.

## 4 PRELIMINARY TESTS

In this section we will present some test results of both artificial and real datasets. First we demonstrate the concept in detail using a basic test graph. Second, we use a real dataset to demonstrate that different exploration paths result in congruent layouts.

### 4.1 Concept verification

The first test run is based on a graph of the form of a Venn diagram for three sets (see Figure 6). It contains three node rings that overlap at the center. This graph is small yet complex enough to test the correctness of our approach. Its structure allows, on the one hand, the extraction of three overlapping patches – the rings – and ensures, on the other hand that their layouts overlap only very little while having excellent matching error scores.

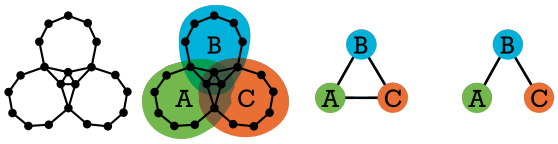



Figure 6: From left to right: The Venn diagram (1) is split into three overlapping subgraphs (2). These form a patch graph (3) with 3 patches and 3 edges which is reduced to a tree (4) to enable a stable interactive exploration.

We create the patch graph and compute a spanning tree. Using the tree, we then merge one patch after another in coordination with the interactive exploration component. The green patch is shown first and thus forms the root patch for rendering and remains as it is (Figure 7 left). The blue patch is flipped, rotated and translated to the bottom of the green, minimizing the matching error between the green and the blue patch. The common nodes are then merged, creating the layout in Figure 8 center). In the next step, the orange patch is aligned with the blue, already aligned patch. This results in a total transformation (Figure 8 right) of about 180° for the orange patch. Lastly, the node positions are unified where necessary and merged for the final, visible graph.

 ISBN 978-80-86943-74-9

Figure 7: Individual layouts of the three patches of the Venn diagram graph.

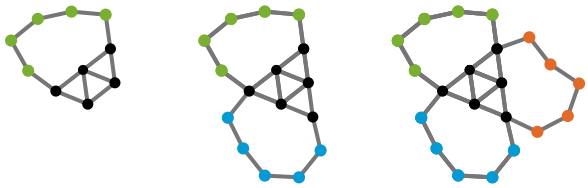


Figure 8: Initially, only the green layout is visible (left). In the next step, the blue patch is matched against the green patch (center). Finally, the orange patch is matched to the blue patch (right).

The second test we performed was with a pair of star-shaped subgraphs which has been extracted from a real dataset. The layout of both subgraphs is strongly affected by the high degree of the central nodes. The intersection of the node sets contains only four elements, but both star nodes are included. This leads to a significant overlap of the patches, but the star patterns are still visible (Figure 9).

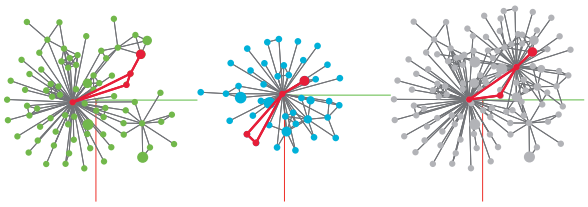


Figure 9: Two star-shaped patches (green, blue) are stitched together forming a single graph layout (gray). The common nodes (red) that form the base for the matching are the only nodes that are distorted. Although the central nodes are in both sets, both patches are recognizable in the stitched layout.

In a similar dataset, two star-shaped graphs have been extracted again, but this time, they intersect only at their boundaries.

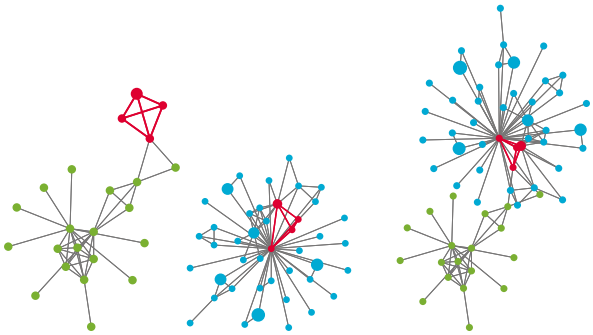


Figure 10: Two star-shaped patches (green, blue). A clear outlier region in the green patch leads to a clear separation in the stitched layout (right).

As can be seen in Figure 10, the patterns are stitched with only very little deformation of the original shapes. More importantly, the nodes that form the connection between the two clusters are clearly distinguishable in the stitched layout.

## 4.2 Exploration independence

The claim of this paper is to create a deterministic layout which is independent of the exploration path. We test this hypothesis by navigating through several clusters of a larger graph in different order and compare the generated layouts.

The dataset for this test is a network graph from the medical domain [GCV+07] with roughly 1.5k nodes, 5.5k edges. Our clustering algorithm created 77 layout patches. The size of the graph features a fair amount of complexity while still being visually comprehensible when viewed as a whole (Figure 11). We used a force-directed layout of the whole dataset to display the exploration path as ground truth and compare the results. With only one single parameter – the number of iterations – the *chinese whispers* algorithm [Bie06] appeared to be a good choice for the clustering of this graph.

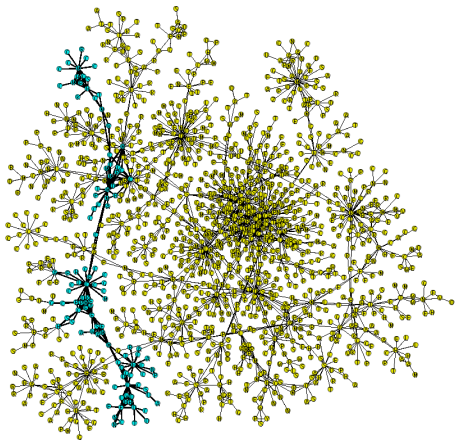


Figure 11: The explored graph part in the global layout

Several different explorations have been performed to verify the validity of our approach. They all started at different points exploring the same clusters, but in different orders. As can be seen in the teaser figure on the first page, the resulting stitched layouts are congruent. The construction of two exemplary stitched layouts is depicted in Figure 12 and Figure 13, respectively.

# 5 CONCLUSION & OUTLOOK

In this paper we presented a new approach that aims to create dynamic graph layouts which are independent of the exploration path. It works independent of specific layout algorithms and thus also works for highly dynamic force-directed layout algorithms. When the user explores large graphs with dynamic views, new nodes

 ISBN 978-80-86943-74-9

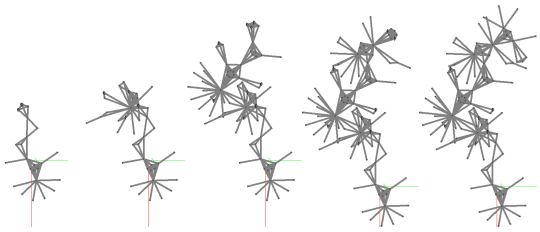


Figure 12: The first exploration through five clusters in the order 1, 2, 3, 4, 5.

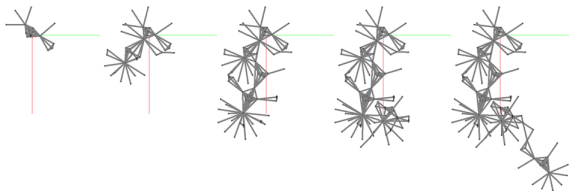


Figure 13: In the second run, the clusters were explored in the inverse order resulting in a congruent layout.

are typically added in the proximity of existing, linked nodes.

This approach is thus highly dependent on the exploration path – the layout can look very different even for very similar explorations. Our method overcomes this limitation with techniques from the computer vision domain where image stitching is used to merge multiple photographs with overlapping areas into a larger image. In analogy to that, our method uses pre-computed layout patches that are sewn together in deterministic order. Consequently, the resulting layout is stable, independent of the user's exploration path and will, thus, always look the same. In contrast to many other dynamic graph layout algorithms, a fair amount of computational effort can be pre-computed which increases the interactivity and reduces the workload at runtime. Being able to work with different layout algorithms for different patches makes it also very versatile.

Compared to conventional layout methods, the additional computational effort is also rather small. The cost of layout computation is increased by the factor of nodes that exist in multiple matches. Runtime costs are limited to the creation a $2 \times 2$ cost matrix and its decomposition which has a constant running time [MHTG05].

The layout stitching method we presented sees the subgraph as a disconnected point cloud and merges the patches without respect to the topological structure. Closely related to that, it also ignores the points that are not in the intersection of the two nodes sets. As a result, two layouts could be aligned so that the disjoint parts overlap as well which is undesired.

We assume that more sophisticated approaches for the computation of the patch overlaps could mitigate this problem and improve the stitching quality. This includes the use of the graph topology metrics such as connectivity to find and include the best-fitting nodes. An ideal strategy would include nodes that emphasize certain visual features of the cluster layout to make the structure memorable. We are convinced that the interpretation of layouts as images features a plethora of concepts and approaches just waiting to be transferred and applied to graph layouts.

## 6 REFERENCES


[AHB87] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698 –700, sept. 1987.

[AMA07] D. Archambault, T. Munzner, and D. Auber. TopoLayout: Multilevel Graph Layout by Topological Features. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):305 –317, march-april 2007.

[APP11] D. Archambault, H. Purchase, and B. Pinaud. Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 17(4):539 –552, april 2011.

[Bie06] Chris Biemann. Chinese Whispers: an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, TextGraphs-1, pages 73–80, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[BL07] Matthew Brown and David G. Lowe. Automatic Panoramic Image Stitching using Invariant Features. *International Journal of Computer Vision*, 74:59–73, 2007.

[BM12] Ulrik Brandes and Martin Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In Marc Kreveld and Bettina Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes in Computer Science*, pages 99–110. Springer Berlin Heidelberg, 2012.

[DMS+08] T. Dwyer, K. Marriott, F. Schreiber, P. Stuckey, M. Woodward, and M. Wybrow. Exploration of networks using overview+detail with constraint-based cooperative layout. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1293 –1300, nov.-dec. 2008.

[EHK+04] Cesim Erten, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler,

and Gary Yee. GraphAEL: Graph Animations with Evolving Layouts. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 98–110. Springer Berlin Heidelberg, 2004.

[ELMS91] Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. Preserving the mental map of a diagram. *Proceedings of COMPUGRAPHICS*, 91(9):24–33, 1991.

[FT04] Y. Frishman and A. Tal. Dynamic Drawing of Clustered Graphs. In *IEEE Symposium on Information Visualization (InfoVis '04)*, pages 191–198, 2004.

[FT08] Y. Frishman and A. Tal. Online Dynamic Graph Drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727 –740, july-aug. 2008.

[GBPD05] Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic Graph Drawing of Sequences of Orthogonal and Hierarchical Graphs. In János Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 228–238. Springer Berlin Heidelberg, 2005.

[GCV+07] Kwang-Il Goh, Michael E. Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-Lázló Barabási. The Human Disease Network. *Proc. of the National Academy of Sciences USA*, 104(21):8685–8690, 2007.

[HEL05] Xiaodi Huang, Peter Eades, and Wei Lai. A framework of filtering, clustering and dynamic layout graphs for visualization. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38*, ACSC '05, pages 87–96, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[HEW98] Mao Lin Huang, Peter Eades, and Junhu Wang. Online Animated Visualization of Huge Graphs using a Modified Spring Algorithm. *Journal of Visual Languages & Computing*, 9(6):623–645, 1998.

[Kob12] Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.

[LPP+06] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, BELIV '06, pages 1–5, New York, NY, USA, 2006. ACM.

[MCH+09] Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. Topology-aware navigation in large networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2319–2328, New York, NY, USA, 2009. ACM.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless Deformations Based on Shape Matching. *ACM Trans. Graph.*, 24(3):471–478, July 2005.

[MPWG12] K. Marriott, H.C. Purchase, M. Wybrow, and C. Goncu. Memorability of Visual Features in Network Diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2477–2485, dec. 2012.

[MSDK12] T. May, M. Steiger, J. Davey, and J. Kohlhammer. Using Signposts for Navigation in Large Graphs. *Computer Graphics Forum*, 31(3 pt. 2):985–994, 2012.

[Nor96] Stephen C. North. Incremental layout in dynadag. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 409–418, London, UK, UK, 1996. Springer-Verlag.

[SP08] Peter Saffrey and Helen Purchase. The "mental map" versus "static aesthetic" compromise in dynamic graphs: a user study. In *Proceedings of the ninth conference on Australasian user interface - Volume 76*, AUIC '08, pages 85–93, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[vHP09] Frank van Ham and Adam Perer. Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. *IEEE Transactions on Visualization and Computer Graphics*, 15:953–960, 2009.

[vHP12] Frank van Ham and Adam Perer. Integrating Querying and Browsing in Partial Graph Visualizations. *IBM Technical Report 12-01*, 2012.

[WRR03] Michael Wall, Andreas Rechtsteiner, and Luis Rocha. Singular Value Decomposition and Principal Component Analysis. *A practical approach to microarray data analysis*, pages 91–109, 2003.

 ISBN 978-80-86943-74-9