UNIVERSITY OF LATVIA
Institute of Mathematics and Computer Science

KĀRLIS FREIVALDS

ALGORITHMS FOR VISUALIZATION OF GRAPH-BASED STRUCTURES

Doctoral Thesis

Advisor:
Assoc. professor, Dr. math.
P. Ķikusts

Riga - 2004

Contents

Summary

The thesis is formed as a collection of 5 related papers [10, 11, 12, 13, 24] about visualization of graph-based structures. Several methods and algorithms for automated creation of graph-like diagram layouts are presented in the thesis, as well as algorithms for interactive diagram input and modification.

Graph-based structures arise in many fields where it is natural to represent objects and their relations in form of a graph. For visualization purposes graph-based structure is represented by a graph-like diagram, which is mainly a graph but additional constructions, such as hyper edges and relation representation by nesting, are allowed. (Fig. 1).

The goal of the thesis was to investigate the visualization of graph-like information by means of easy readable diagrams where object and relation features relevant for the user, such as closeness, connectedness, symmetry, are visually easily perceptible in the drawing. In case of large and complex graph-like structures creating diagrams by hand is time consuming, hence algorithmic tools have to be created that simplifies and automates the visualization process. For this purpose theoretic research was carried out and, based on the results, methods and effective algorithms were developed for automatic layout of diagram elements.

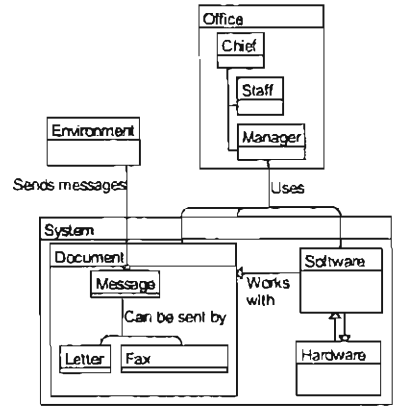


Fig. 1. Drawing of a complex graph-like structure.

The foundation of graph-based structure visualization is graph drawing discipline [2, 3], which develops and explores algorithms for automatic creation of graph drawings. Graph drawing is a fairly new branch but is developing rapidly due to increase of the systems complexity and their describing graph sizes when creating a drawing by hand becomes complicated. The main tasks for drawing a graph-like diagram are:

- Automatic creation of the drawing;
- Automatic modification of the drawing, when its underlying structure has been changed;
- Interactive input and exploration of the diagram

Automatic creation of the drawing of the diagram is the most widely used operation because often we have a combinatorial description of data in form of a graph, which is obtained automatically from database or constructed by some algorithm, and to obtain easily perceptible understanding about the essence of the data, a drawing of the diagram must be created. If acquired data change over time, for example, due to changing the structure of the database, then automatic modification of the diagram has to be applied to ensure that the drawing contains the latest data.

Sometimes not the data is primary, but the drawing created by the user. Then convenient diagram input and editing are important. Afterwards, the necessary data is generated from the created graph-like diagram.

But no matter how drawing of the diagram is obtained, it is important that the diagram is easily readable and shows those properties of the graph that are essential for specific application. For example, in Fig. 2 and 3 two drawings of the same graph are shown. In Fig. 3 it can be easily noticed that the graph consists of two independent parts, which is not obvious from Fig. 2. Besides, by adding labels, graphics attributes e.g. colors, icons or line styles to the graph, we can tell much more about our data as in other ways. In general, several aesthetics [2, 3] in graph drawing are established that tell when a drawing would be easily readable. They are:

- Edges are short;
- Edges are as straight as possible;
- The crossing number of edges is small;
- Node symbols intersect neither each other nor edges.

Each drawing of a graph should respect these aesthetics as much as possible, although some layout styles may treat them differently. For example, in the case of orthogonal edges, the straightness of edges is measured as the number of bends. Also it is often not enough to ensure that diagram symbols do not intersect and ensuring some minimum spacing between them is required.

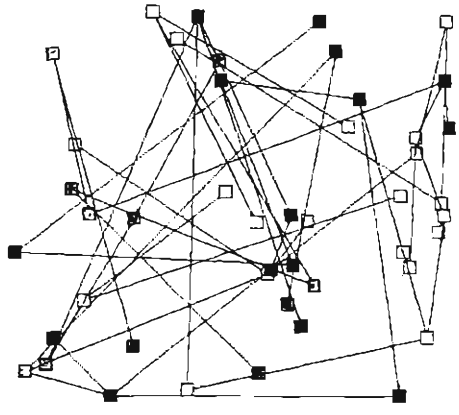


Fig. 2. Graph before layout.

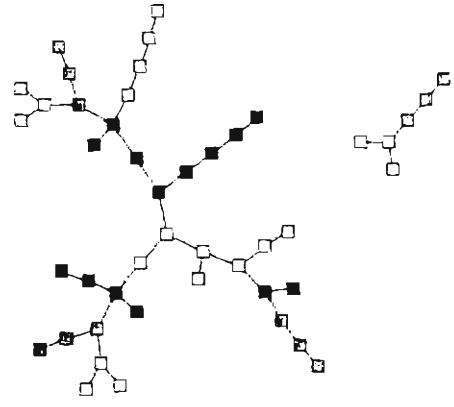


Fig. 3. Graph after layout.

Individual graph drawing algorithms are usually grouped in a larger module, which provides full spectrum of general automatic layout and diagram editing operations. Then, depending on the application, a specific tool is made based on such module, which is adopted for the required tasks.

In Latvia the graph drawing tradition began in 1990 with the development of a business modeling tool GRADE. This tool supports description of business processes with graph-like diagrams, which are entered by the user. The user feedback shows that this tool is the winner in the field of interactive diagram editing even now what is achieved by successfully crafted graph layout algorithms.

Later, graph drawing module "*Graphical Diagramming Engine*" was developed on basis of the obtained knowledge with substantial participation of the author of the thesis, which implemented emerged algorithms in a new quality. Based on this module several graph drawing tools were developed. One of them is *LinkViewer*, which graphically displays links between various Web pages, automatically laying out the resulting graph. Here the size of the graph can reach several thousands of nodes and edges, hence the effectiveness of the algorithm is highly important.

Now, the worlds leading graph drawing tool is acknowledged to be "*Tom Sawyer Layout*", which provides the widest set of features in the highest quality. The author of this thesis participates in the development of this tool now.

Diagram Deformation

Important concept arising when the user interactively works with the diagram is *mental map* [21, 4]. It is the shape of the diagram that is remembered or imagined by the user. To ensure that the diagram is still recognizable after its modification, it is necessary that all diagram operations keep this mental map. But the drawing tool should help to keep the diagram tidy, ensuring non-overlapping of the diagram symbols.

To support interactive work with the diagram, the following basic operations are necessary:

- Insertion of a diagram symbol;
- Modification of a diagram symbol;
- Removal of a diagram symbol.

To insert a symbol in the place pointed by the user ensuring the non-overlapping aesthetic, a free space must be obtained where to put this symbol. Similarly, deleting a symbol should remove the empty space by compaction of the diagram. Modification of a symbol can be implemented as its removal followed by insertion of the modified symbol.

We propose to formulate all these actions in a unified way as an optimization problem – minimize the deformation of the diagram ensuring minimum distance requirements [13]. By dividing the process into two passes, one for vertical segments and one for the horizontal, and expressing the problem mathematically, deformation can be measured as the total squared drift of diagram symbols in the corresponding direction but minimal distances can be represented by an obstacle graph (Fig. 4). A weight equal to the required amount of the free space is assigned to each obstacle graph edge. Then by assigning a variable x_k to each segment, the optimal deformation optimization problem can be formulated with the following mathematical programming task:

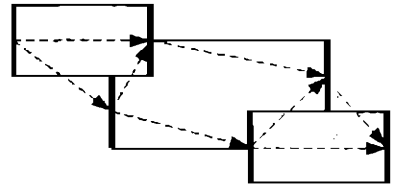


Fig. 4. Obstacle graph of vertical segments.

$\min \sum_k (x_k - c_k)^2$, where c_k is the old coordinate of the k-th segment

subject to $x_j - x_i \geq d_{ij}$ for each obstacle graph edge (i, j) with weight d_{ij}.

The new coordinates x_k of the segment give the shape of the deformed diagram, which is close to the initial one and where the minimal distances are ensured. In a similar way, by adding new terms to the optimization function, it is possible to ensure other relevant diagram features, for example to minimize node sizes.

In the chapter "Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing" of the thesis an efficient algorithm for solving this problem is proposed and several other applications of this optimization method including rectangle packing and intersection removal [16], diagram correction and compaction [13], providing free place for labels [17], as well as solving some layout sub-problems [2, 14] are discussed. Due to the well-behaved structure of the constraints we achieve very fast running time of the optimization algorithm suitable not only for diagram layout but also for real-time application in interactive editing.

Layout

Depending on the application there can be several characteristics that can be shown in the drawing of a graph. These characteristics are often grouped and a separate layout style is created for revealing the chosen features. Let us consider the main layout styles in the graph drawing module "*Tom Sawyer Layout*". They are: hierarchical (Fig. 5), orthogonal (Fig. 6), symmetrical (Fig. 7) and circular (Fig. 8).

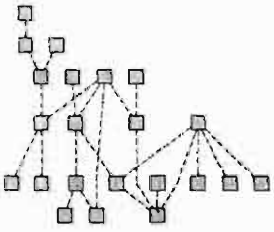


Fig. 4. Hierarchical layout style.

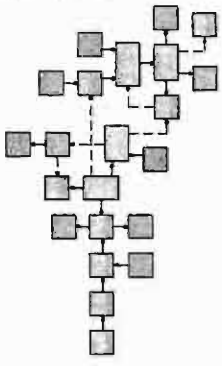


Fig. 5. Orthogonal layout style.

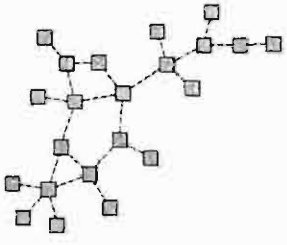


Fig. 6. Symmetrical layout style.

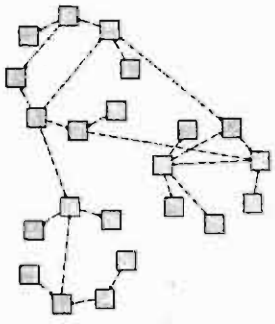


Fig. 7. Circular layout style.

Although visually the styles are different, the layout is produced in a unified way by combining the following layout phases:

- Node layout;
- Edge layout;
- Label placement;
- Component packing.

Depending from the selection of the algorithm in each phase a specific layout style is obtained.

Node Layout

The task for node layout is to place the nodes on the plane corresponding to the specifics of the given layout style. This is the first and the most important step of the graph layout since node position impact on the layout quality is the greatest.

Let us discuss one of the mentioned node placement algorithms – the circular layout for which the author has developed a new clustering algorithm. According to this style the graph is divided into node subsets called clusters where nodes of each cluster are placed on a circle such that edge crossings between them are minimized. Then a cluster graph is formed by merging all nodes of each cluster into one node of the size of whole cluster. This graph is laid out using some other layout style, symmetric, for instance, and then the new nodes are replaced back by the cluster drawings to obtain the final layout.

For the layout to be pleasing, the most important part is obtaining an adequate graph clustering. Circular layout is supposed to be used when the size of the graph is too large to comprehend and its visual complexity should be reduced by clustering. This idea leads us to the following clustering task: dividing nodes into clusters so that the number of clusters would be significantly less than the number of nodes in the initial graph but the cluster graph would keep the properties of the input graph as much as possible.

For the purpose of "Tom Sawyer Layout" the author has implemented the following algorithm, which is based on recursive bisection:

```
(1)   Create one cluster from all nodes of the graph;
(2)   Calculate the desired cluster size as a square root of the
        number of nodes;

(3)   while (size of the maximal cluster greater than the desired)
(4)   {
(5)     Find the maximal cluster;
(6)     Divide it into two using the ratio-cut algorithm;
(7)   }
```

To get good results it is important to use a balanced cut for dividing the cluster [19]. One possibility of such cut is ratio-cut [25] for which the author has developed an efficient algorithm [10]. The algorithm is based on solving a certain mathematical optimization problem, which is deeply analyzed in the chapter "A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning" of the thesis. In case when all arising sub-problems can be solved exactly, the obtained algorithm gives a factor 2 approximation, while the approximation factor for the best previously known algorithms [23, 18, 20, 19, 25] can achieve $\log n$.

The presented clustering algorithm, which uses the described ratio-cut algorithm, gives much better results than other clustering algorithms that were used for layout purposes.

Edge Layout

When the nodes are positioned, the next step is edge layout. The edges are placed to connect the corresponding nodes and satisfying the qualities of the required layout style.

Three kinds of edge layout styles are popular – orthogonal (Fig. 9), polyline (Fig. 10) and curved line [3] (Fig. 11). The most natural but algorithmically hardest obtainable and less explored is curved edge style [5, 14, 15]. In the chapter "Curved edge routing" an original algorithm is proposed for producing edges of such kind what is achieved by expressing layout aesthetics with a cost function and using mathematical optimization to search a line, which best matches these aesthetics [12].

At first, a cost function is constructed, which maps a number to each point on the plane how costly it is for an edge to go through this point. This function is expressed as a sum of several terms each corresponding to some of the graph objects – a node or already placed edge. In case of a node this term function is infinite inside the node and rapidly decreasing outside it to ensure a certain distance between the routable edge and the node. In case of an edge the function is of some finite magnitude and again decreasing outside the edge. In this way the aesthetics of low edge crossings and separation are modeled.

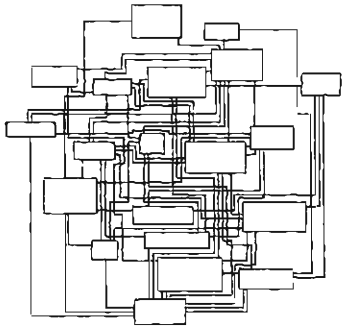


Fig. 8. Orthogonal edge style.

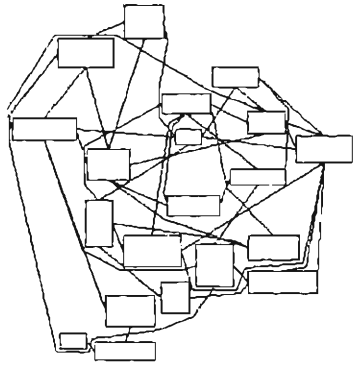


Fig. 9. Polyline edge style

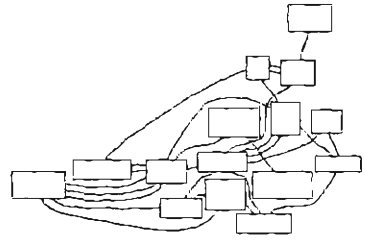


Fig. 10. Curved edge style

When the cost function is created, the edge shape is calculated as a line with the smallest total cost, which connects the respective nodes. It is accomplished by solving a differential equation using numerical methods. By using the modern "*Level set*" method [22], solution is obtained using moderate computational resources. The obtained edge shape is very smooth and natural, like drawn by hand.

In general, such approach expressing aesthetics with a cost function allows to route also orthogonal and polyline edges, however, simpler and faster routing algorithms for such edges exist [2, 8].

Component Packing

Graph layout algorithms should be able to deal with any kind of input graphs, including those consisting of several non-connected parts called components (Fig. 13). But several basic layout algorithms, for example, symmetric layout [2, 3] requires the graph to be connected. In such cases the graph is divided into components before layout, which are laid out independently using the required algorithm and then compactly packed together. Traditionally for this task each component is represented by a rectangle for which efficient packing algorithms are known [1, 6, 7, 9] but often such approximation is too rough and packing density is decreased.

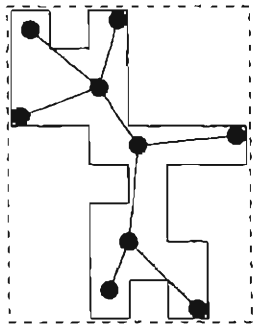


Fig. 12. Polyomino approximation.

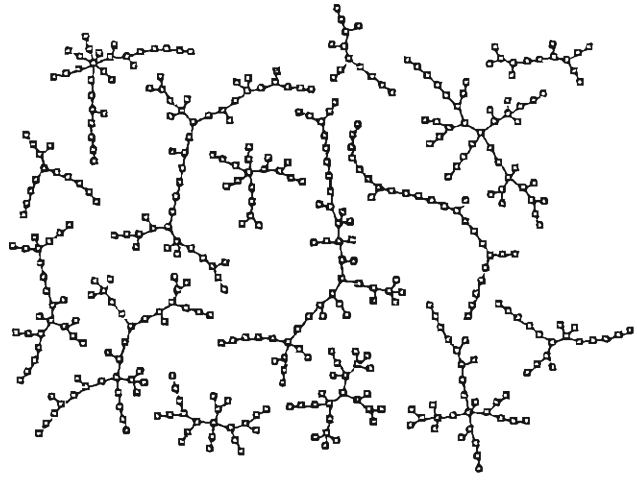


Fig. 11. Component packing using polyomino based algorithm.

In the chapter "Disconnected Graph Layout and the Polyomino Packing Approach" a packing method is developed where each component is approximated with a polyomino (Fig. 12), which encloses a drawing of a component much tighter. Algorithms for polyomino packing in different scenarios are presented: minimizing the total used area, ensuring the aspect ratio of the packing area and packing in pages of fixed size [11]. The results of the proposed algorithm are demonstrated on random forest graphs (Fig. 13) and compared to other known algorithms. The polyomino packing algorithm delivers significantly better packing density not only for components but also for packing rectangles. The running time is slightly worse than for rectangle packing algorithms but is acceptable for practical graphs comprising up to several thousands of components.

Validation of the Algorithms for Layout of Gene Expression Graphs

Based on the "Graphical Diagramming Engine" graph drawing module, gene expression graph visualization tool was developed [24] where the obtained drawings allowed to identify several new gene features. Important graph features that need to be visualized here are node in- and out-degrees as well as connected components of the

graph. Mostly the hierarchical layout algorithm was exploited, since it clearly shows the nodes with high in- and out-degrees by placing them in the opposite sides of the drawing. Also the symmetric layout was used to produce pleasing drawings for presentation. The components were placed with the polyomino packing algorithm discussed in the thesis. Deeper information about the used algorithms and results are provided in the chapter of this paper "Building and analysing genome-wide gene disruption networks".

References

1. B. S. Baker, E. G. Coffman, R. S. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846--855, November 1980.
2. G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis. Graph Drawing, Prentice Hall, 1999.
3. Battista, G., Eades, P., Tamassia, R., & Tollis, I.G. Algorithms for Drawing Graphs: an Annotated Bibliography. *Computational Geometry: Theory and Applications*, Vol. 4, 1994, pp. 235–282
4. S. Bridgeman, R. Tamassia. Difference metrics for interactive orthogonal graph drawing algorithms, – Proc. of Graph Drawing '98, *Lecture Notes in Computer Science*, vol. 1547, 1998, pp. 57–71.
5. C. C. Cheng, C. A. Duncan, M. T. Goodrich, S. G. Koburov. Drawing Planar Graphs with Circular Arcs. Symp. on Graph Drawing GD'99, *Lecture Notes in Computer Science*, vol.1731 , pp.117–126, 1999.
6. E. G. Coffman, M. R. Garey, D. S. Johnson, R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808--826, November 1990.
7. E. G. Coffman and P. W. Shor. Packings in two dimensions: Asymptotic average-case analysis of algorithms. *Algorithmica*, 9:253--277, 1993.
8. D.P. Dobkin, E.R. Gansner, E. Koutsofios, Stephen C. North. Implementing a General-Purpose Edge Router. Symp. on Graph Drawing GD'97, *Lecture Notes in Computer Science*, vol.1353 , pp.262–271, 1998.
9. U. Dogrusoz. Algorithms for layout of disconnected graphs. In *Proc. of the Fifth International Conference on Computer Science and Informatics (CS\&I 2000)*, vol. 1, pages 539--542, 2000.
10. K. Freivalds, A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning, Workshop on Experimental and Efficient Algorithms (WEA 2003), *Lecture Notes in Computer Science* vol. 2647, pp. 134–147. Springer-Verlag, 2003.
11. K. Freivalds, U. Dogrusoz, and P. Kikusts, Disconnected Graph Layout and the Polyomino Packing Approach, Proc. of Graph Drawing 2001, *Lecture Notes in Computer Science*, vol. 2265, pp. 378-391, Springer-Verlag, 2002.
12. K. Freivalds, Curved Edge Routing, Proc. of the Symposium on Fundamentals of Computation Theory 2001, *Lecture Notes in Computer Science* vol. 2138, pp. 126–137.

13. K. Freivalds, P. Kikusts Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing. *Proc. of the Latvian Academy of Sciences*, Section B, Vol. 55 (2001), No. 1, pp. 43–51.
14. E. R. Gansner, E. Koutsofios, S. C. North, K-P. Vo. A Technique for Drawing Directed Graphs, – *TSE* vol. 19, no. 3, 1993, pp. 214–230.
15. M. T. Goodrich, C. G. Wagner. A Framework for Drawing Planar Graphs with Curves and Polylines. Symp. on Graph Drawing GD'98, *Lecture Notes in Computer Science*, vol.1547 , pp.153–166, 1998.
16. K. Hayashi, M. Inoue, T. Masuzawa, H. Fujiwara. A layout adjustment problem for disjoint rectangles preserving orthogonal order, – Proc. of Graph Drawing '98, *Lecture Notes in Computer Science*, vol. 1547, 1998, pp. 183–197.
17. G. W. Klau, P. Mutzel. Combining graph labeling and compaction, – Proc. of Graph Drawing '99, *Lecture Notes in Computer Science*, vol. 1731, 1999, pp. 27–37.
18. P. Klein, S. Plotkin, C. Stein, E. Tardos, Faster approximation algorithms for unit capacity concurrent flow problems with applications to routing and sparsest cuts, *SIAM J. Computing*, 3(23) (1994), pp. 466–488.
19. T. Leighton, S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, vol 46 , No. 6 (Nov. 1999), pp. 787 – 832.
20. D. W. Matula, F. Shahrokhi, Sparsest Cuts and Bottlenecks in Graphs. *Journal of Disc. Applied Math.*, vol. 27 (1990), pp. 113–123.
21. K. Misue, P. Eades, W. Lai, K. Sugiyama. Layout adjustment and the mental map, – *Journal of Visual Languages and Computing*, vol. 6, 1995, pp. 183–210.
22. J.A. Sethian. Level Set Methods. Cambridge University Press, 1996.
23. F. Shahroki, D. W. Matula, The maximum concurrent flow problem. *Journal of the ACM*, vol. 37, pp. 318–334, 1990.
24. J. Rung, T. Schlitt, A. Brazma, K. Freivalds, J. Vilo Building and analysing genome-wide gene disruption networks. *Bioinformatics* 2002 Oct;18 Suppl 2:S202-210. European Conference on Computational Biology (ECCB 2002).
25. Y. C. Wei, C. K. Cheng, Ratio Cut Partitioning for Hierarchical Designs. *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 911–921, July 1991.

PROCEEDINGS OF THE LATVIAN ACADEMY OF SCIENCES. Section B, Vol. 55 (2001), No. 1 (612), pp. 43–51.

OPTIMUM LAYOUT ADJUSTMENT SUPPORTING ORDERING CONSTRAINTS IN GRAPH-LIKE DIAGRAM DRAWING

Kārlis Freivalds and Paulis Ķikusts

Institute of Mathematics and Computer Science, University of Latvia, Raiņa bulv. 29, Rīgā LV-1459, LATVIA;
e-mail: {karlisf,paulis}@mii.lu.lv

Communicated by Jānis Bārzdiņš

We propose an optimisation-based technique for layout operations ensuring flexible and convenient interactive editing of a wide class of graph-like diagrams. Diagrams may be very complex, containing nested nodes, textual labels on connection paths, and branched structures of paths. Layout operations rely on a mental map preserving optimum layout adjustment via solving a quadratic programming problem subject to ordering constraints. For this purpose, we have developed a highly efficient mathematical programming method. This method also enables to solve several related optimisation problems, such as optimum placement of vertices into layers, or horizontal coordinate assignment for layered vertices, which were considered to be difficult in practice.

Key words: *Graph drawing, mathematical programming, mental map.*

INTRODUCTION

Graph-like diagrams are graph-based pictorial models that indicate the interrelationships of elements of various structures. Graph-like diagrams are widely used in many areas to describe information and its structure, for example, to describe the connections between enterprises for the development of specifications, or for program code representation (Martin and McClure 1988; Booch *et al.*, 1999).

An important aspect for users is diagram visualisation, i.e. drawing them in a readable way. Such a process is called diagram layout. A layout of a diagram can be created interactively by the user, or automatically by a program. The interactive drawing approach (Di Battista *et al.*, 1999) has led to a concept of a *mental map*, i.e. the general view of the diagram imagined by the user (Bridgeman and Tamassia, 1998; Di Battista *et al.*, 1999; Misue *et al.*, 1995). The mental map of a diagram needs to be preserved during the layout process in order to ensure the user's control and understanding. Thereby all changes to the diagram always have to be minimised, so the optimisation approach rises in a natural way along with the notion of a mental map.

The concept of a mental map, together with optimisation questions, has been much studied in research literature. Misue *et al.* (1995) discusses the problem of preservation of the mental map. The authors propose several models to make the concept of a mental map more precise: orthogonal ordering, proximity relations, and topology. Hayashi *et al.* (1998) further develops the approach to avoid the intersections among rectangles. In a paper by Bridgeman and Tamassia (1998), formalisation of the notion of a mental map is performed, and the differences between layouts in various aspects are expressed mathematically: distance, proximity, orthogonal ordering, shape and topology. Other authors (He and Marriott, 1997) use mathematical programming, including quadratic, to preserve the mental map in an interactive layout when repeated modifications occur.

STATEMENT OF THE PROBLEM

Our graph-like diagrams are combinatorial structures consisting of three principal kinds of elements: nodes, relations among nodes, and labels. Each element is represented by a corresponding geometrical object. A layout of a diagram is an arrangement of these geometrical objects on the plane (Figure 1).

Nodes are basically represented as two-dimensional symbols, most commonly by upright rectangular boxes or circles. Here, we will use only rectangular boxes.

Relations are represented by: (1) *paths*, i.e. single rectilinear polylines connecting symbols that represent the associated nodes, Figure 1a; (2) *forks*, i.e. branched structures of rectilinear polylines, Figure 1b; and (3) *inclusions*, i.e. placing one node symbol inside another, Figure 1c.

A fork is modelled by introducing a point-shaped object called a *support*, which is the common endpoint of the paths forming a fork (Figure 1b).

Labels are text fields represented by upright rectangles, and are categorised into node labels and path labels. Node labels are placed inside the nodes on the specially assigned mar-

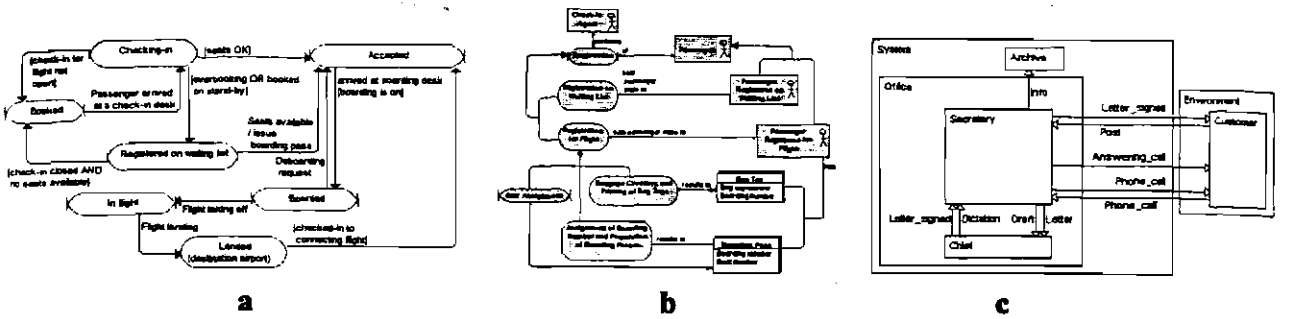


Fig. 1. Simple diagram (a), diagram with forks (b), and diagram with nested nodes (c).

gins. Path labels are placed near their lines in an understandable way to identify which label belongs to which line.

Since we allow a wide range of geometrical representations of relations, our layouts cover the full spectrum, from drawings of simple graphs (Fig. 1a) up to UML diagrams (Booch *et al.*, 1999) (Fig. 1b), including essentially generalised K. Sugiyama's and K. Misue's compound graphs (Sugiyama and Misue, 1991) (Fig. 1c). Figure 2 depicts this entire spectrum.

The task of diagram layout is to represent the information of diagrams in an easily perceptible way (Ding and Mateti 1990; Protsko *et al.*, 1991). Accordingly, a correct layout must satisfy the following natural geometric constraints:

(C1) node rectangles are not smaller than a minimum size,

(C2) path lines have no common segments,

(C3) the minimum distance $\delta > 0$ is guaranteed between nonintersecting segments of geometrical objects,

(C4) path labels neither overlap each other, nor node contours, nor path lines,

(C5) node contours do not cross each other,

(C6) path lines do not intersect node contours unless forced by inclusions.

We allow variable size node rectangles for several reasons. At first, we have to be able to draw graphs of degree much higher than four (Di Battista *et al.*, 1999; Miriyala *et al.*, 1993). Also, editing node labels or putting one node inside another one could cause changes in node sizes. Similarly, inserting new nodes or path labels between the paths contacting the same node may require changing its size.

To create the layout of a diagram interactively or automatically, we go through several relatively independent stages. The main stages are node layout, path routing, and label assignment. At each stage, we provide a correct intermediate layout which preserves a common mental map. Modifications to the layout, while preserving the mental map, are done by optimum layout adjustment via solving two quadratic programming problems, separately for the horizontal and the vertical directions, subject to the ordering constraints corresponding to each direction.

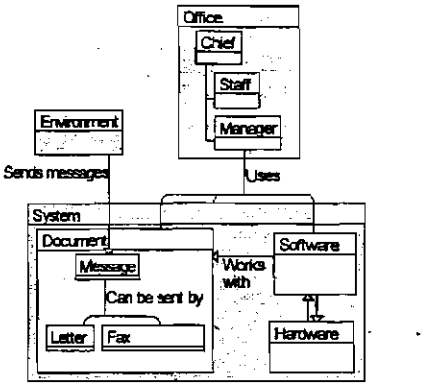


Fig. 2. Complex diagram.

This approach conforms to several important ideas proposed in the literature. First, layout adjustment requires the objects to move or to stretch (Miriyala *et al.*, 1993). Furthermore, an optimum adjustment involves mathematical programming (including integer, linear, and quadratic) that is widely used in graph drawing (He and Marriott, 1997, Di Battista *et al.*, 1999). Recently, Di Battista, Didimo *et al.* (1999) and Klau and Mutzel (1999) also elaborated related concepts and touch ours in some basic points.

The deviation of the layout from the intended mental map can be measured by a function to be minimised. Our function comes from the concepts of distance metrics (Bridgeman and Tamassia, 1998) and position constraints (Di Battista *et al.*, 1999). Minimisation is done subject to *ordering constraints*. Di Battista *et al.* (1999) showed how ordering constraints can be used in layered drawings for horizontal coordinate assignment. However, when discussing the use of a quadratic programming approach, the authors warn that the solution requires considerable computational resources, even if the ordering constraints form an acyclic graph. Such constraints are also described by Gansner *et al.* (1993) for finding optimum layering by integer programming. Below, we will show that our technique, which is based on the gradient projection method (Minoux, 1986), allows us to solve these problems spending quite moderate computational resources.

A quadratic programming algorithm is the principal part of our procedure called *Normalise*, which ensures a correct intermediate layout while not destroying the common mental

map at each layout creation stage. *Normalise* is our backbone operation and is discussed below.

LAYOUT NORMALISATION

When modifying a diagram, the user inserts new nodes or paths, adds path labels, or changes the geometrical attributes of diagram elements. Without difficulty, all of these actions can be accomplished keeping the constraints C4, C5, and C6 satisfied, yet the remaining constraints may be violated. To satisfy the constraints C4, C5, and C6, new nodes and path labels have to be represented by zero-size rectangles (i.e. points) located in the desired positions (the bold dots in Fig. 3a).

The *Normalise* operation ensures that the constraints C1 to C6 are satisfied, including minimal distances between newly routed paths, and minimal sizes of elements preserving the initial mental map. For newly added nodes and path labels the *Normalise* operation assigns correct sizes to these elements. Also correct node inclusions are ensured.

Preservation of the mental map means for us minimisation of the total distance between the new and the old locations of the diagram elements while keeping their ordering undisturbed. Since our diagram elements are represented by upright rectangles or rectilinear polylines, the layout geometry consists only of vertical and horizontal line segments. Therefore, all layout modification operations we can do separately: first, for the vertical layout segments (Fig. 3a), then, for the horizontal layout segments (Fig. 3b). Note that after processing the vertical segments, the nodes and path labels represented by points become horizontal segments due to minimum size requirements (Fig. 3b).

Let us consider more closely the case of vertical line segments. In this case, it is necessary to find only the x-coordinate of each segment. The objective is to assign the x-coordinates to the segments in a way that a chosen cost function attains its minimum, taking certain constraints into account. The basic constraints here are minimum horizontal distance requirements.

To retain the general view of the given layout, the ordering of segments is predetermined in some sense. The main idea is a segment *obstacle* relation, which is derived from segment visibility: segment b is an obstacle of segment a if:

- projections of the extended segments of a and b on the vertical axis overlap,

- the abscissa of a is smaller than the abscissa of b,

- there is no segment c between a and b such that c is an obstacle for a, and b is an obstacle for c.

Here, for the given segment, the *extended segment* is a segment which is obtained from the given one by extending its both ends by $\frac{\delta}{2}$ (refer to constraint C3), if the given segment is of non-zero length. Such a relation allows for point-shaped objects to slide freely among other diagram objects, while path endpoints remain enclosed between the corresponding node sides.

The obstacle relation defines the *obstacle graph* of the segments. The obstacle graph is planar, therefore, its edge number is small. Moreover, the edges of the obstacle graph may be acyclically directed from left to right, thus defining the basic ordering of layout segments.

The basic ordering does not represent the complete ordering information. Some additional relations have to be added to ensure correct ordering for newly inserted nested nodes that are represented by single points. In order to guarantee constraint C2, a special procedure is called for overlapping path segments to avoid unnecessary path crossings.

After a complete segment ordering is determined, it is also represented by a graph. Besides ordering information, we include additional arcs into this graph: from the left segment of every node to its right segment, thus ensuring minimum node size constraint C1. We call the graph obtained the *constraint graph*. Like the obstacle graph, the constraint graph is acyclic, and also small.

We have found that layout optimality may be expressed via a quadratic optimisation problem:

minimise $F(x_1, x_2, \ldots x_n)$
subject to $x_j - x_i \geq d_{ij} \geq 0$,

where $x_1, x_2, \ldots x_n$ are the x-coordinates of the segments, and the pairs (i, j) are the arcs of the constraint graph.

The function F is built to minimise the changes of the layout, and in the most usual form is a sum comprising summands of two kinds corresponding only to diagram nodes.

To minimise the node drift, we introduce the summands $(\frac{x_l + x_r}{2} - x_c)^2$, where for each node, x_l, x_r are the abscissas of its left and right segments, and x_c is a constant denoting the abscissa of its old centre. To minimise the node size, F also comprises the summands of the form $w(x_r - x_l)^2$.

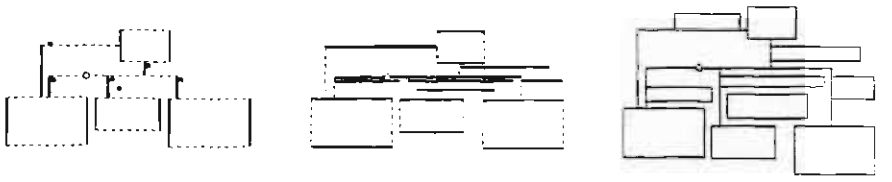


Fig. 3. Vertical layout segments (a), horizontal layout segments (b), and all layout segments (c). The bold dots are zero-sized nodes and labels, the support is depicted as a circle.

The weighting factor w should be chosen in an appropriate way. The value 10 seems adequate.

After the minimisation problem has been solved, the diagram is recalculated for the new locations of the vertical segments (Fig. 3b).

Analogously, the layout is processed in the vertical direction (Figs. 3b, 3c).

OPTIMISATION TECHNIQUE

As described above, we must deal with functions in the form

$$F(x) = \sum_k L_k^2(x), \tag{1}$$

where $L_k(x)$ denotes some linear function depending on an n-dimensional point $x = (x_1, x_2, \ldots x_n)^T$. We need to minimise F subject to the inequality

$$Ax \geq d, \tag{2}$$

where each row r of the $m \times n$ matrix A comprises only two non-zero elements -1 and $+1$ in columns i_r and j_r respectively, and all the pairs (i_r, j_r) form an acyclic graph.

We have chosen the gradient projection method (Minoux, 1986) as a theoretical background for solving this quadratic programming problem.

The solution is found in two stages. First a feasible starting point x_0 satisfying the inequality $Ax_0 \geq d$ is searched. If such a point exists, then our problem obviously has a solution.

Lemma 1. *The set of feasible points is non-empty.*

Proof. Let us number the vertices of the constraint graph topologically, and let d_{max} be the maximum component of m-tuple d. By setting $x_i = i \cdot d_{max}$ we satisfy the condition (2). ■

After the starting point is found, iterations are performed in order to find the solution. At each iteration, the current point x is changed so that F decreases.

We have to distinguish two major cases: whether the inequality (2) is strong or not.

Case $Ax > d$.

In this case, the point x is strongly inside the feasible area and we may shift x in the direction of the steepest descent $g = (-\nabla F(x))^T$.

We find two scalar values: τ_1 minimising the function $f(\tau) = F(x + g \cdot \tau)$, $\tau \geq 0$, and

$$\tau_2 = \max (\tau \geq 0 \mid A \cdot (x + g \cdot \tau) \geq d).$$

Finding both τ_1 and τ_2 is easy, because $f(\tau)$ is a quadratic function and (2) is reduced to m linear inequalities of one variable.

Then x has to be changed to $x + g \cdot \min(\tau_1, \tau_2)$.

Case $Ax \geq d$, and equality holds for at least one dimension.

In this case, the point x is on the border of the feasible area and we must shift x along the border in the direction which is the projection p of g onto the border.

To calculate p, let us introduce a new $m_0 \times n$ matrix A_0 as the submatrix of A consisting of those rows of A for which strong equalities in (2) take place. Let d_0 be the corresponding subcolumn of d. We call the corresponding subgraph of the constraint graph the *active constraint graph* and denote it by G_0.

Lemma 2. *All vertices of every connected subgraph of G_0 have mutually equal corresponding projection components.*

Proof. From the choice of A_0 we have $A_0 x = d_0$, and for an arbitrary shift y along the border defined by A_0, we have $A_0 \cdot (x + y) = d_0$, too. Hence

$$A_0 y = 0, \tag{3}$$

and consequently $A_0 p = 0$.

The last equality means that, for an arbitrary row of A_0, we have $p_i = p_j$, i.e. all arcs of G_0 have equal projection components for both ends. The required statement follows immediately. ■

Lemma 3. *Let S be the index set of vertices of an arbitrary maximum connected subgraph of G_0, and, as stated above, all its vertices have the same projection component p_S. Then*

$$p_S = \frac{1}{|S|} \sum_{k \in S} g_k .$$

Proof. As p is the projection of g, $g - p$ is perpendicular to all directions y along the border. Because of (3), $g - p$ can be expressed as some linear combination of rows of A_0, i.e. taking an appropriate m_0-tuple u

$$g - p = A_0^T u. \tag{4}$$

The k-th row in the last equality is $g_k - p_k = \sum_{i=1}^{m_0} a_{ik} u_i$ where a_{ik} denotes an element of A_0.

We have $\sum_{k \in S} (g_k - p_k) = \sum_{k \in S} \sum_{i=1}^{m_0} a_{ik} u_i = \sum_{i=1}^{m_0} u_i \sum_{k \in S} a_{ik}$ and, since S includes none or both ends of G_0's arcs, $\sum_{k \in S} a_{ik} = 0$, because each row of A_0 comprises exactly two non-zero elements -1 and $+1$. Hence $\sum_{k \in S} (g_k - p_k) = 0$, and $\sum_{k \in S} g_k = \sum_{k \in S} p_k = |S| \cdot p_S$. ■

Lemmas 2 and 3 show that p is calculable from g in a very simple way.

After p is calculated, we need to distinguish two other cases.

Case $p \neq 0$.

In this case, as in the case $Ax > d$ we find two scalar values: τ_1 minimising the function $f(\tau) = F(x + p\cdot\tau)$, $\tau \geq 0$, and $\tau_2 = \max(\tau \geq 0 \mid A\cdot(x + p\cdot\tau) \geq d)$. And then change x to $x + p\cdot\min(\tau_1, \tau_2)$.

Case $p = 0$.

In this case we need to change the matrix A_0 or stop the iterations. Because of the convexity of our optimisation problem, the Kuhn-Tucker conditions (Minoux, 1986) allow us to distinguish between the two cases.

From (4), we have

$$g = A_0^{\mathrm{T}} u. \tag{5}$$

The Kuhn-Tucker conditions mean that if there exists u satisfying (5) and

$$u_i \leq 0,\ i = 1, 2, \ldots m_0, \tag{6}$$

then the optimum is reached.

Lemma 4. *Let all vertices of G_0 be partitioned into two disjoint subsets V and $\bar{V}$ in such a way that all arcs joining V and $\bar{V}$ go from V to $\bar{V}$ thus forming a directed cut separating V and $\bar{V}$. Let $\bar{S}$ be the index set of vertices of $\bar{V}$. If the cut is* positive, *i.e.* $\sum_{k\in\bar{S}} g_k > 0$, *then every u satisfying* (5) *violates* (6). *Also, g is directed inside the feasible area relative to its border defined by those rows of A_0, which correspond to the arcs of the cut, and the projection of g onto the feasible area's border defined by the other rows of A_0 is not equal to* 0.

Proof. Let C be the index set of rows of A_0 corresponding to the arcs of the cut.

Since each row of A_0 comprises exactly two non zero elements (–1 and +1) that indicate the endpoints of the arc corresponding to the row, and since only arcs of the cut have exactly one (marked with +1) endpoint belonging to $\bar{V}$, we have $\sum_{k\in\bar{S}} a_{ik} = \begin{cases} 1, \text{if } i \in C \\ 0, \text{otherwise} \end{cases}$.

Hence, as u satisfies (5), $\sum_{k\in\bar{S}} g_k = \sum_{k\in\bar{S}} \sum_{i=1}^{m_0} a_{ik} u_i = \sum_{i=1}^{m_0} u_i \sum_{k\in\bar{S}} a_{ik} = \sum_{i\in C} u_i$, and $\sum_{i\in C} u_i > 0$, because of the definition of a positive cut (refer to the statement of this Lemma). Obviously, for some i $u_i > 0$, i.e. (6) does not hold.

To prove that g is directed inside the feasible area relatively to its border defined by those rows of A_0 which correspond to the arcs of the cut, we show that there exists u satisfying (5) such that $u_i \geq 0$ for all $i \in C$.

Assume first that G_0 is connected and our cut is a minimum cut, i.e. any proper subset of its arcs does not form a cut. In such a case, there exists a spanning tree in G_0 that includes exactly one arc from our cut. We remove from G_0 all arcs of the cut except the one of the spanning tree, and we remove from A_0 the corresponding rows, thus obtaining the graph and the matrix A_0'. Besides, there exists such a $(m_0 - |C| + 1)$-tuple u' for which $g = A_0'^{\mathrm{T}} u'$.

In the graph G_0' the vertex sets V and $\bar{V}$ are still separated by a positive directed cut. Hence, by the same argument as for u, the unique component of u' corresponding to the cut is positive. It is easy to see that the required u is obtainable from u' by setting all missing components to 0.

In the case when G_0 is disconnected or our cut is not a minimum one, those parts of the cut, which are minimum cuts, must be examined separately in each maximum connected subgraph of G_0.

Finally, let us show that the projection of g onto the feasible area's border defined by those rows of A_0 which do not correspond to the arcs of our cut is not equal to 0.

Denote by G_1 the graph obtained from G_0 after removing all arcs of the cut. Some of G_1's maximum connected subgraphs constitute the part $\bar{V}$. Let S_j ($j = 1, \ldots$) be the vertex index sets of these subgraphs: $\bar{S} = S_1 \cup \ldots$. Since S_j are mutually disjoint and $\sum_{k\in\bar{S}} g_k > 0$, some of the sums $\sum_{k\in S_j} g_k$ must be different from 0. Recalled Lemma 3, this means the required property of the projection of g. ■

Lemma 5. *If for every directed cut separating V and $\bar{V}$ in G_0, $\sum_{k\in\bar{S}} g_k \leq 0$ holds, then there exists u satisfying* (5) *and* (6).

Proof. Let us extend G_0 by adding two new vertices s and t, and by adding additional arcs from s to all vertices with $g_k \geq 0$, and from all vertices with $g_k < 0$ to t. We are about to pass a flow through the extended graph. The capacity of the original arcs is set to ∞, and the capacity of all arcs adjacent to s or t is the value $|g_k|$ corresponding to the second end of the arc.

There exists a flow with a value $g^+ = \sum_{g_k\geq 0} g_k$. To prove this, we have to verify the well-known Ford-Fulkerson condition: the total capacity c_{in} for all ingoing arcs of every set $V\cup\{t\}$ must be at least $g+$, where V is a subset of the vertices of G_0.

If at least one arc from G_0 goes into V, then $c_{in} = \infty > g^+$.

In the opposite case, G_0 has a directed cut separating V and $\bar{V}$.

Let S and $\bar{S}$ be the index sets of vertices from V and $\bar{V}$ respectively, and

$$g_S^+ = \sum_{k\in S, g_k\geq 0} g_k,\ g_{\bar{S}}^- = \sum_{k\in\bar{S}, g_k<0} g_k.$$

It is clear that $g^+ = g^+_S + g^+_{\bar{S}}$, and, by the condition of the Lemma to be proved, $g^+_S + g^-_{\bar{S}} \le 0$.

Since only arcs going into $V \cup \{t\}$ are adjacent to s or t, $c_{in} = g^+_S - g^-_{\bar{S}} = g^+ - g^-_{\bar{S}} \ge g^+$.

Thus, a flow with a value g^+ exists and gives the values $\varphi_i \ge 0$, $i = 1, 2, \ldots m_0$ to arcs of G_0.

Let $In(k)$ and $Out(k)$ denote the index sets of ingoing and outgoing arcs of the kth vertex of G_0. It holds that $In(k) = \{i \mid a_{ik} > 0\}$, $Out(k) = \{i \mid a_{ik} < 0\}$.

It is easy to see that for our flow we have:

$$g_k = \sum_{i \in Out(k)} \varphi_i - \sum_{i \in In(k)} \varphi_i = \sum_{i: a_{ik} < 0} \varphi_i - \sum_{i: a_{ik} > 0} \varphi_i = \sum_{i=1}^{m_0} a_{ik} \cdot (-\varphi_i).$$

Hence $g = A_0^{\mathrm{T}}(-\varphi)$, where $\varphi = (\varphi_1, \varphi_2, \ldots \varphi_{m_0})^{\mathrm{T}}$. ■

Lemmas 4 and 5 show how to distinguish, in the case $p = 0$, between changing the active constraint graph or stopping the iterations. If there exists a positive directed cut, iterations must be continued beforehand removing the rows corresponding to the arcs of the cut from A_0.

Testing of the existence of such a cut is the most complex part of our optimisation method. Fortunately, the question is well-studied and can be solved by the maximum flow technique (Cormen *et al.*, 1990).

The method may be made significantly faster due to a very clear geometric background of the problem. Indeed, according to Lemmas 2 and 3, when we shift the current point x to its new position, each maximum-connected-component of the active constraint graph moves as a rigid body. There is no need to move all components simultaneously by the vector $g \cdot \min(\tau_1, \tau_2)$. Any direction where F decreases is admissible. We can take components one by one and shift them in a direction which decreases the function. If two components touch each other, we merge them.

The outline of our algorithm follows:

(1) Shift and merge components of the active constraint graph while possible;

(2) Calculate a positive cut;

(3) If such a cut exists, remove its arcs from the active constraint graph and continue with (1).

Furthermore, we can eliminate costly flow computations by maintaining a spanning tree in each component. At each merge, we update the tree by adding one active arc between the two components. The necessary cut of the tree can be calculated in linear time.

APPLICATION EXAMPLES

The main and the most important application example is diagram normalisation. To measure the time complexity of our optimisation method, we generated a series of realistic-looking diagram examples randomly in the following way. We take N random upright rectangles representing diagram nodes. Placing them randomly, they may intersect (Fig. 4a). To obtain a correct diagram, intersections must be eliminated. This task is solved by our technique, giving the initial node layout (Fig. 4b). Next, we add N random independently routed paths. Independent routing may generate path segments that violate minimum distance requirements, which are then corrected by the *Normalise* operation (Fig. 4c). As the last step we add path labels, freeing the required space using one more *Normalise* operation (Fig. 4d). In all steps, the mental map of the initial rectangle positions is preserved.

Basically, the preservation of the mental map is expressed as minimisation of node drift subject to obstacle graph requirements. We can use a different model as well, for example preserving the orthogonal ordering as discussed previously (Misue *et al.*, 1995; Hayashi *et al.*, 1998). In our technique, we only add arcs between adjacent rectangles (with respect to the ordering) to our constraint graph. Figure 5 shows the rectangle intersection elimination while preserving the orthogonal ordering applied to the same starting position (Fig. 4a).

Tables 1 and 2 show the performance of the C++ implementation of our optimisation method running on a PENTIUM 120Mhz computer. The average data from ten examples is taken. Table 1 reflects diagram processing illustrated in Fig-

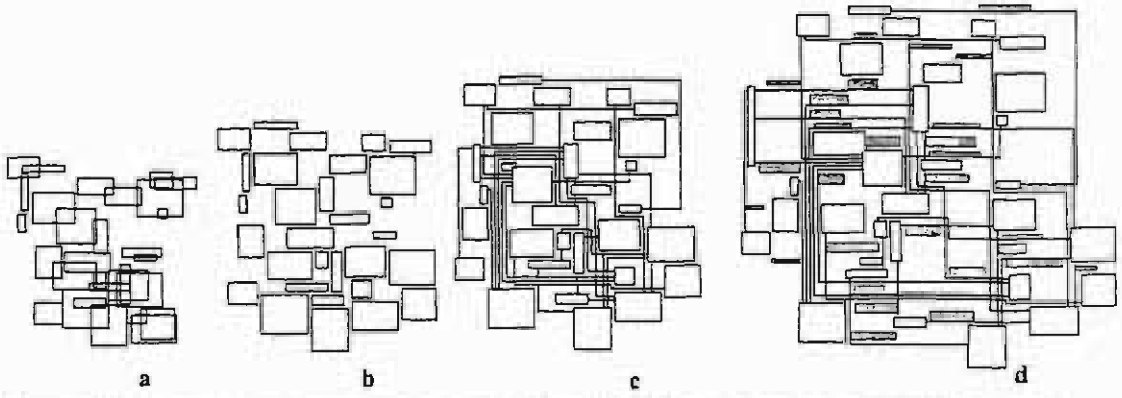


Fig. 4. Initial rectangles (a), rectangles after intersection elimination (b), normalised random paths (c), and random size path labels (d).

Table 1

DIAGRAM PROCESSING

Intersection elimination					Path routing						Labeling					
n	I_x	T_x	I_y	T_y	n_x	I_x	T_x	n_y	I_y	T_y	n_x	I_x	T_x	n_y	I_y	T_y
1000	9	0.1	18	0.3	5610	37	3.1	5545	37	2.9	6610	10	1.3	6545	21	2.2
2000	11	0.3	29	0.9	13140	60	12.7	13001	46	9.2	15140	9	2.7	15001	26	6.9
3000	16	0.6	38	1.6	21616	66	21.7	21427	58	18.1	24616	10	4.4	24427	34	14.2
4000	16	0.9	48	2.7	30899	82	38.3	30640	75	33.0	34899	11	7.3	34640	36	22.6

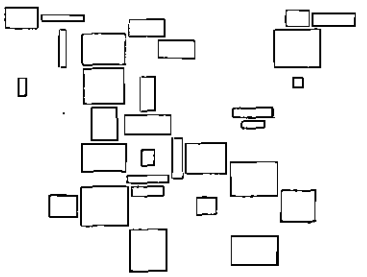


Fig. 5. Rectangles of Fig. 4a after intersection elimination preserving orthogonal ordering

Table 2

RECTANGLE INTERSECTION ELIMINATION

	Obstacle ordering				Orthogonal ordering			
n	I_x	T_x	I_y	T_y	I_x	T_x	I_y	T_y
1000	8	0.1	17	0.2	17	0.2	24	0.3
2000	12	0.3	26	0.7	27	0.7	37	1.0
4000	16	1.0	46	2.8	43	2.4	60	3.4
8000	24	3.0	80	10.6	74	8.8	105	13.0
16000	38	8.6	145	35.2	123	31.6	193	51.5

ure 4. Table 2 shows elimination of rectangle intersections on larger datasets in two cases: (1) preserving the orthogonal ordering, and (2) only with the obstacle graph approach. The segment count (n), iteration count (I) and time in seconds (T) are given in the x and y directions separately.

The processing time obtained in our experiments can be expressed as $\sim n^p$, where $1.5 < p < 2$ depending on the problem type. Such time performance is quite good in practice since our diagrams usually contain up to several thousand segments, and the method is fast enough.

DISCUSSION

Our optimum layout adjustment technique was developed to handle graph-like diagrams of complex structure. Our *normalisation* concept is very powerful and allows gradual diagram layout creation in several stages. An independent path routing, followed by normalisation, is very convenient because the node layout stage does not have to consider the paths in great extent. We can process the most complex path structures, including forks, after the nodes have been laid out. The known algorithms today either do not deal with forks or demand some simplifying conditions. For example, the algorithm given by Seemann (1997) requires forks to form an acyclic graph. Handling forks as *supports* removes such limitations.

Other operations based on our normalisation concept are layout *correction* and *compaction*. Correction is a *Normalise*-like procedure that leads to the constraints C1...C6 being satisfied. We only have to replace all nodes by zero-sized rectangles and calculate a correct constraint graph. Compaction is another analogue of the *Normalise* procedure. It reduces the distance between nodes by minimising some other cost function.

Generally, our proposed optimisation technique is usable in automatic layout, when diagrams are reduced to graphs. We combine two algorithms that lay out undirected and directed graphs, respectively.

Since we use a grid in the automatic graph layout, we have to notice that with practically good approximation, our optimisation technique solves the corresponding integer programming problem by simply rounding off the real-valued solution. More importantly, we are not restricted to a quadratic function, since the same algorithm also handles a linear one. Besides, in the integer linear case we obtain the exact result because of the simplicity of our constraints.

An undirected graph is laid out on the grid, repeatedly moving each vertex to a free gridpoint nearest to the barycentre of its neighbours. To ensure free gridpoints near the desired place, we expand the layout, time after time, by compacting it and inserting empty rows and columns.

A directed graph is laid out conventionally in a layered structure (Gansner *et al.*, 1993, Di Battista *et al.*, 1999). If the graph contains cycles, then the number of edges going upward is minimised and temporally reversed, thus obtaining an acyclic graph. First, vertices are placed into layers, and then ordered inside them to minimise edge crossings. In both cases, our optimisation technique is involved in the following way.

In accordance with (Gansner *et al.*, 1993, Di Battista *et al.*, 1999), optimum placement of vertices into layers minimises

the total vertical extent of all edges, i.e. the sum of differences between layer numbers of edge vertices. Taking our temporary acyclic graph as the constraint graph and requiring the vertical extent of each edge to be at least 1, we immediately obtain an integer linear programming problem, which is solved as mentioned above.

Further, vertices are ordered inside layers according to their neighbour barycentres. To keep the vertices separated, we perform the *Normalise* operation. After the vertex order is determined, we assign the final horizontal coordinates, minimising the total edge length squares as suggested by Di Battista *et al.* (1999). We sum only squares of the horizontal extent of the edges, and minimise this sum subject to constraints induced by extremely simple linear vertex ordering inside each layer. Despite the caution given by Di Battista *et al.* (1999), our optimisation technique does not require considerable computational resources and solves the minimisation problem efficiently.

Another particularly important stage of layout creation is path label assignment. Maintaining textual labels on paths is a hard problem managed only by a few systems (Ķikusts and Ručevskis, 1996; Dogrusoz *et al.*, 1998; *http://www.gradetools.com/*). Our approach essentially facilitates the labelling problem by separating it into two independent and technically simpler subproblems: (1) looking for free places, and (2) deforming the layout if there is not enough space. Having good initial label positions, the *Normalise* procedure provides their correct size while preserving the initial mental map, thus obtaining quite pretty path labelling (*http://www.gradetools.com/*).

ACKNOWLEDGEMENTS

The authors would like to thank the GRADE development group at Institute of Mathematics and Computer Science of University of Latvia for enabling them to participate in an interesting collaborative project, and particularly Kārlis Podnieks and Andris Zariņš.

The work was supported by the Latvian Council of Science under Grant 96.0247 and by ESPRIT project No 23287 ADDE.

REFERENCES

Booch, G., Rumbaugh, J., Jacobson, I. (1999) *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts. xxii, 482 p.

Bridgeman, S., Tamassia, R. (1998) Difference metrics for interactive orthogonal graph drawing algorithms. In: *Graph Drawing. Proceedings of the 6th International Symposium, GD '98, Montreal, Canada, August 13–15, 1998*. Whitesides, S. H. (ed.). *Lecture Notes in Computer Science*, Vol. **1547**, 57–71.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. (1990) *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts. xvii, 1028 p.

Received September 25, 2000.

Di Battista, G., Didimo, W., Patrignani, M., Pizzonia, M. (1999) Orthogonal and quasi-upward drawings with vertices of prescribed size. In: *Graph Drawing. Proceedings of the 7th International Symposium, GD '99, Stirin Castle, Czech Republic, September 15–19, 1999*. Kratochvil, J. (ed.). *Lecture Notes in Computer Science*, Vol. **1731**, 297–310.

Di Battista, G., Eades, P., Tamassia, R., Tollis, I. G. (1999) *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, New Jersey. xi, 397 p.

Ding, C., Mateti, P. (1990) A framework for the automated drawing of data structure diagrams. *IEEE Trans. Software Eng.*, **16**, 543–557.

Dogrusoz, U., Kakoulis, K. G., Madden, B., Tollis, I. G. (1998) Edge labeling in the Graph Layout Toolkit. In: *Graph Drawing. Proceedings of the 6th International Symposium, GD '98, Montreal, Canada, August 13–15, 1998*. Whitesides, S. H. (ed.). *Lecture Notes in Computer Science*, Vol. **1547**, 356–363.

Gansner, E. R., Koutsofios, E., North, S. C., Vo, K-P. (1993) A Technique for Drawing Directed Graphs. *IEEE Trans. Software Eng.*, **19**, 214–230.

Hayashi, K., Inoue, M., Masuzawa, T., Fujiwara, H. (1998) A layout adjustment problem for disjoint rectangles preserving orthogonal order. In: *Graph Drawing. Proceedings of the 6th International Symposium, GD '98, Montreal, Canada, August 13–15, 1998*. Whitesides, S. H. (ed.). *Lecture Notes in Computer Science*, Vol. **1547**, 183–197.

He, W., Marriott, K. (1997) Constrained graph layout. In: *Graph Drawing. Proceedings of the Symposium on Graph Drawing, GD '96, Berkeley, California, USA, September 18–20, 1996*. North, S. (ed.). *Lecture Notes in Computer Science*, Vol. **1190**, 217–232.

Ķikusts, P., Ručevskis, P. (1996) Layout algorithms of graph-like diagrams of GRADE Windows graphic editors. In: *Graph Drawing. Proceedings of the* Symposium on Graph Drawing, GD '95, Passau, Germany, September 20–22, 1995. Brandenburg, F. J. (ed.). *Lecture Notes in Computer Science*, Vol. **1027**, 361–364.

Klau, G. W., Mutzel, P. (1999) Combining graph labeling and compaction. In: *Graph Drawing. Proceedings of the 7th International Symposium, GD '99, Stirin Castle, Czech Republic, September 15-19, 1999*. Kratochvil, J. (ed.). *Lecture Notes in Computer Science*, Vol. **1731**, 27–37.

Martin, J., McClure, C. (1988) *Structured Techniques: The Basis for Case*. Prentice Hall, Englewood Cliffs, New Jersey. xxviii, 776 p.

Minoux, M. (1986) *Mathematical Programming: Theory and Algorithms*. Wiley, Chichester [W. Sussex]; New York. xxviii, 489 p.

Miriyala, K., Hornick, S. W., Tamassia, R. (1993) An incremental approach to aesthetic graph layout. In: *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering: CASE '93*. Lee, H.-Y., Reid, T. F., Jarzabek, S. (eds.). IEEE Computer Society Press, Los Alamitos, CA, pp. 297–308.

Misue, K., Eades, P., Lai, W., Sugiyama, K. (1995) Layout adjustment and the mental map. *J. Visual Lang. Computing*, **6**, 183–210.

Protsko, L. B., Sorenson, P. G., Tremblay, J. P., Schaefer, D. A. (1991) Towards the automatic generation of software diagrams. *IEEE Trans Software Eng.*, **17**, 10–21.

Seemann, J. (1997) Extending the Sugiyama algorithm for drawing UML class diagrams: towards automatic layout of object-oriented software diagrams. In: *Graph Drawing. 5th International Symposium, GD '97, Rome, Italy, September 18–20, 1997, Proceedings*. Di Battista, G. (ed.). *Lecture Notes in Computer Science*, Vol. **1353**, 415–424.

Sugiyama, K., Misue, K. (1991) Visualization of structural information: automatic drawing of compound digraphs. *IEEE Trans. Syst. Man Cybern.*, **21**, 876–892.

IZVIETOJUMA OPTIMĀLA SAKĀRTOŠANA PIE SECĪBAS IEROBEŽOJUMIEM GRAFVEIDA DIAGRAMMU ZĪMĒŠANĀ

Piedāvāts matemātiskās optimizācijas paņēmiens interaktīvām izvietošanas operācijām, ar kuru palīdzību elastīgi un ērti ar datoru ir konstruējamas un rediģējamas grafveida diagrammas. Diagrammas var būt ļoti sarežģītas un saturēt ģeometriski iekļautus blokus, uzrakstus pie savienojumu līnijām, kā arī zarotus savienojumus. Izvietošanas operācijas balstītas uz izvietojuma optimālu sakārtošanu, saglabājot lietotāja iecerēto diagrammas vispārējo izskatu. To dara, atrisinot *kvadrātiskās programmēšanas uzdevumu pie secības ierobežojumiem*. Šim nolūkam darbā izstrādāta īpaši efektīva matemātiskās programmēšanas metode, un tā dod iespēju atrisināt arī virkni radniecīgu optimizācijas uzdevumu, tādu kā optimāls virsotņu izvietojums pa slāņiem vai horizontālo koordinašu piešķiršana slāņu virsotnēm, kuri līdz šim bija uzskatīti par praktiski grūti risināmiem.

A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning

Kārlis Freivalds

University of Latvia
Institute of Mathematics and Computer Science
Raina blvd. 29, LV-1459, Riga, Latvia
Karlis.Freivalds@mii.lu.lv

Abstract. We propose a new method for finding the minimum ratio-cut of a graph. Ratio-cut is NP-hard problem for which the best previously known algorithm gives an $O(\log n)$-factor approximation by solving its dually related maximum concurrent flow problem. We formulate the minimum ratio-cut as a certain nondifferentiable optimization problem, and show that the global minimum of the optimization problem is equal to the minimum ratio-cut. Moreover, we provide strong symbolic computation based evidence that any strict local minimum gives an approximation by a factor of 2. We also give an efficient heuristic algorithm for finding a local minimum of the proposed optimization problem based on standard nondifferentiable optimization methods and evaluate its performance on several families of graphs. We achieve $O(n^{1.6})$ experimentally obtained running time on these graphs.

1 Introduction

Balanced cuts of a graph are hard computation problems important both in theory and practice. Ratio-cut is the most fundamental one since most of the others including minimum quotient cut, minimum bisection, multi-way cuts can be easily approximated using it [13, 20]. Also several other important approximation algorithms like crossing number and minimum cut linear arrangement are based on the ratio-cut [13]. Ratio-cut has many practical applications, most important being VLSI design, clustering and partitioning [23, 14, 1].

Since ratio-cut is a NP-hard problem [15] we must seek for approximation algorithms to solve it in practically reasonable time. Many purely heuristic algorithms were developed [23, 25, 21, 8] most of them relying on simulated annealing, spectral methods or iterative movement of nodes from one side of the partition to the other. A common idea exploited by several authors [25, 2, 9, 21, 22] to improve their quality is using multi-scale graph representation usually obtained by edge contraction. At first a partition at the coarsest scale is obtained and then refined to a more detailed one by one of mentioned algorithms. Although these algorithms may perform well in practice no optimality bounds are known for them.

The best previously known algorithm with proven optimality bounds finds an $O(\log n)$-factor approximation, where n is the number of nodes in the graph. It is based on reduction of the ratio-cut problem to the multi-commodity flow problem, which can be solved with polynomial time linear programming methods. Unfortunately this method is not practical since the resulting linear program is of quadratic size of the number of nodes in the graph and cannot be solved efficiently. Then, approximation algorithms [16, 17, 12, 24] were discovered for the multi-commodity flow problem itself making this approach usable in practice. Several heuristic implementations [16, 11, 24] are based on this idea, some of them quite effective and practical. The most elaborate one [11] can deal with up to 100000-node graphs in reasonable time.

In this paper we propose a new way of finding the minimum ratio-cut of a graph. We construct a nondifferentiable optimization problem whose minimum solution equals the minimum ratio-cut value and use nonlinear programming methods to search for it. Since the problem is non-convex, we may find only a local minimum. However, we show that any strict local minimum gives us a factor of 2 approximate cut. For that purpose we introduce a notion of locally minimal ratio cut for which no subset of nodes taken from one side of the cut and moved to the other side decrease the cut value. We establish one-to-one correspondence between strictly locally minimal cuts and strict local minima of the proposed optimization problem.

The reduction of a NP-hard discrete problem to a continuous one is not a novel idea. For example the maximum clique problem of a graph can also be stated as an optimization problem [6] and numerical optimization methods for finding the optimum may be used. However for the maximum clique problem no optimality bounds of a local minimum are known. To show that our method is practical we present an efficient heuristic algorithm for finding a local minimum of the proposed problem, which is based on the standard methods of nondifferentiable optimization and analyze its performance on several families of graphs. With the proposed method we can find a good partition of a 200000-node graphs in less than one hour.

2 Problem Formulation

We are dealing with an undirected graph $G = (V, E)$, where V is its node set and E is its edge set. The nodes of the graph are identified by natural numbers from 1 to n. Each node i has a weight $d_i = 0$, and each edge (i, j) has a capacity $c_{ij} = 0$, satisfying the properties $c_{ij} = c_{ji}$, $c_{ii} = 0$. We define $c_{ij} = 0$ when there is no edge (i, j) in G. We assume that there are at least two nodes with non-zero weights. (A, A') denotes a cut that separates a set of nodes A from its complement $A' = V \backslash A$. The capacity of the cut $C(A, A')$ is the sum of edge capacities between A and A'. The ratio of the cut $R(A, A')$ is defined as follows

$$R(A, A') = \frac{C(A, A')}{\sum_{i \in A} d_i \cdot \sum_{i \in A'} d_i} \tag{1}$$

We will focus on finding the minimum ratio-cut i.e. the cut (A, A') with the minimum ratio over all nonempty A, A'.

Definition 1. *A cut (A, A') is locally minimal if for all non-empty $U \subset A, U \neq A$: $R(A, A') \leq R(A \backslash U, A' \cup U)$ and for all non-empty $U' \subset A', U' \neq A'$: $R(A, A') \leq R(A \cup U', A' \backslash U')$. Similarly we call a ratio-cut strictly locally minimal if strong inequalities hold in this definition.*

3 Ratio-Cut as an Optimization Problem

We can assign a variable $x_i \in \mathbb{R}$ to each node i and consider the following optimization problem over x from $\mathbb{R}^n$.

$$\min F(x) = \sum_{i,j \in V} c_{ij} |x_i - x_j| \tag{2}$$

$$\text{subject to } H(x) = \sum_{i,j \in V} d_i d_j |x_i - x_j| = 1, \tag{3}$$

$$\sum_{i \in V} x_i = 0. \tag{4}$$

This optimization problem is equivalent to the ratio-cut problem in the sense described below.

Definition 2. *A characteristic vector x^A for a cut (A, A') is defined such that its components*

$$x_i^A = \begin{cases} a, i \in A \\ b, i \in A' \end{cases}, \text{where} \tag{5}$$

$$a = -\frac{1}{2 \sum_{i \in A} d_i \cdot \sum_{i \in A'} d_i} \cdot \frac{|A'|}{|V|}, b = \frac{1}{2 \sum_{i \in A} d_i \cdot \sum_{i \in A'} d_i} \cdot \frac{|A|}{|V|}.$$

It is straightforward from this definition that for a cut (A, A') x^A satisfies the constraints (3, 4), and $F(x^A) = R(A, A')$.

Definition 3. *For some feasible x and some $p \in \mathbb{R}$ we call the cut (P, P') positional, if $P = \{i | x_i \leq p\}$, $P' = V \backslash P$, and both P and P' are non-empty. The ratio of this cut $R_p(x) = R(P, P')$. For a fixed x we can speak of minimum positional cut $R_{\min}(x)$ over all possible positional cuts obtained for different p:*

$$R_{\min}(x) = \min_{p \in \mathbb{R}} R_p(x).$$

Theorem 1. *For each feasible x^* of (2, 3, 4) $F(x^*) \geq R_{\min}(x^*)$. Also $F(x^*) > R_{\min}(x^*)$ if there are at least two positional cuts with different values.*

Proof. Here we only sketch the main steps of the proof, for details omitted due to space limitation refer to [7]. Let us partition all nodes into three sets U_1, U_2, U_3 as follows.

$$y_1^* = \min_i x_i^*, \quad y_2^* = \min_{i|x_i^*>y_1} x_i^*,$$

$$U_1 = \{i|x_i^* = y_1^*\}, \quad U_2 = \{i|x_i^* = y_2^*\}, \quad U_3 = \{i|x_i^* > y_2^*\}$$

If U_3 is empty there is exactly one positional cut, x^* is in the form of characteristic vector and $F(x^*) = R_{\min}(x^*)$ what concludes this proof, else define

$$y_3^* = \min_{i|x_i^*>y_2} x_i^*.$$

We create a sub-problem of (2, 3, 4) by reducing the original one to a new variable $y = (y_1, y_2, y_3)$. Next, we further restrict it with $y_1 \le y_2 \le y_3$ and can drop the absolute value signs getting:

$$\min F_2(y) = 2 \sum_{i\in U_1, j\in U_2} c_{ij}(y_2 - y_1) + \sum_{i\in U_1, j\in U_3} c_{ij}(y_3 + l_j - y_1) + \sum_{i\in U_2, j\in U_3} c_{ij}(y_3 + l_j - y_2) + K, \quad (6)$$

$$\text{subject to } H_2(y) = 2 \sum_{i\in U_1, j\in U_2} d_i d_j(y_2 - y_1) + \sum_{i\in U_1, j\in U_3} d_i d_j(y_3 + l_j - y_1) + \sum_{i\in U_2, j\in U_3} d_i d_j(y_3 + l_j - y_2) + P, \quad (7)$$

$$|U_1|y_1 + |U_2|y_2 + |U_2|y_2 = 0, \quad (8)$$

$$y_1 \le y_2 \le y_3, \quad (9)$$

where P and K are appropriately calculated constants.

We have obtained a locally equivalent linear program in the sense that for y^* the constraints (7, 8, 9) are satisfied and $F(x^*) = F_2(y^*)$. Also, if we can find a better solution for (6 – 9) we can substitute the result back to the original problem giving a better feasible solution for it. From the linear programming theory it is known that we can find the optimal solution by examining the vertices of the polytope defined by the constraints – in our case that means when one of the inequalities in (9) is satisfied as equality. Let us examine both cases.

In case $y_1 = y_2$ after some calculations we get

$$F_2(y_1, y_1, y_3) = (1 - L)R(U_1 \cup U_2, U_3) + B \quad (10)$$

for appropriate constants L and B. And similarly in case $y_2 = y_3$ we get

$$F_2(y_1, y_2, y_2) = (1 - L)R(U_1, U_2 \cup U_3) + B. \quad (11)$$

We chose the solution y with $y_1 = y_2$ if $R(U_1 \cup U_2, U_3) < R(U_1, U_2 \cup U_3)$, otherwise choose the solution with $y_2 = y_3$. It is evident that if both these ratio costs are non-equal we get a strictly smaller function value. We substitute the solution back into the original problem obtaining a new x. x is a feasible solution of (2, 3, 4) with a smaller or equal function value and the set U_2 merged to U_1 or U_3. We repeat the described process until U_3 is empty. Let us analyze the resulting x and sets U_1 and U_2. We have $F(x) = R(U_1, U_2)$, where (U_1, U_2) is some positional cut of x^* (in fact the minimal one), hence $F(x^*) = F(x) = R_{\min}(x^*)$. If we had some non-equal cuts compared during the process, we have a strict decrease in the function and hence the second statement of the theorem holds. □

Definition 4. *A feasible x^* is a local minimum of (2, 3, 4) if there exists $\varepsilon > 0$ such that $F(x^*) \leq F(x)$ for each x satisfying (3, 4) and $||x - x^*|| < \varepsilon$.*

Definition 5. *A feasible x^* is a strict local minimum of (2, 3, 4) if there exists $\varepsilon > 0$ such that $F(x^*) < F(x)$ for each $x \neq x^*$ satisfying (3, 4) and $||x - x^*|| < \varepsilon$.*

Theorem 2. *Each strict local minimum x^* of (2, 3, 4) is in the form $x^* = x^A$ for some cut (A, A').*

Proof. We need to prove that in a strict local minimum the expression (5) holds for some a and b, the correct values for a and b are guaranteed by the constraints (3) and (4). Assume to the contrary that there are more than two distinct values for the components of x^*. We form the reduced problem like in Theorem 1 obtaining equations $(6-9)$. We are able to do that since U_3 is non-empty due to our assumption. From the linear programming theory a strict local minimum can only be on the vertex of the polytope defined by the constraints $(7-9)$, however in our case y^* cannot be on the vertex since the constraints where equality holds at y^* are less then the number of variables. Consequently, x^* cannot be a strict local minimum for the original problem. Hence our assumption is false and the theorem is proven. □

There is one-to-one correspondence between strictly locally minimal cuts and strict local minimums of (2, 3, 4) as stated in the following theorem.

Theorem 3. *x is a strict local minimum of (2, 3, 4) if and only if x is a characteristic vector of some strictly locally minimal cut (A, A').*

The proof is rather technical and is omitted due to space constraints. See [7] for details.

We shall note that also for each non-strictly minimal ratio cut its characteristic vector x^A gives a local minimum of the function, however the converse is not true. There exist non-strict local minima of (2, 3, 4) with the function value not equal to any locally minimal cut value.

Theorem 4. *The minimum ratio-cut R is equal to the optimum solution of (2, 3, 4).*

Proof. From Theorem 1 $F(x) \geq R_{\min}(x) \geq R$. For the characteristic vector x^A of the minimum ratio cut $F(x^A) = R$. The claim follows immediately. □

Theorem 5. *Each locally minimal cut (A, A') is not grater than two times the minimum ratio cut.*

Proof. Let us denote the optimal cut (B, B'). We form four sets $A \cap B$, $A \cap B'$, $A' \cap B$, $A' \cap B'$ shown in Fig. 1. From Definition 1 the ratio of each of the cuts $C_1 = (A \cap B, V - A \cap B)$, $C_2 = (A \cap B', V - A \cap B')$, $C_3 = (A' \cap B', V - A' \cap B')$, $C_4 = (A' \cap B, V - A' \cap B)$ is at least as large as ratio of $C_{12} = (A, A')$ and $C_{23} = (B, B')$. We form a full graph by taking each of the sets $A \cap B$, $A \cap B'$, $A' \cap B$, $A' \cap B'$ as the nodes of the graph and assign edge capacities as the sum of all edge capacities in the original graph between the corresponding sets. We obtain

$$R_1 = \frac{c_1 + c_4 + c_6}{d_1(d_2 + d_3 + d_4)}, R_2 = \frac{c_1 + c_2 + c_5}{d_2(d_1 + d_3 + d_4)},$$

$$R_3 = \frac{c_2 + c_3 + c_6}{d_3(d_1 + d_2 + d_4)}, R_4 = \frac{c_3 + c_4 + c_5}{d_4(d_1 + d_2 + d_3)},$$

$$R_{12} = \frac{c_2 + c_4 + c_5 + c_6}{(d_1 + d_2)(d_3 + d_4)}, R_{23} = \frac{c_1 + c_3 + c_5 + c_6}{(d_2 + d_3)(d_1 + d_4)}.$$

$$R_1 \geq R_{12}, R_2 \geq R_{12}, R_3 \geq R_{12}, R_4 \geq R_{12}.$$

And the statement of the theorem to be proven translates to $\frac{R_{12}}{R_{23}} \leq 2$. We verified this using symbolical computation in Mathematica 4.0. See [7] for details. □

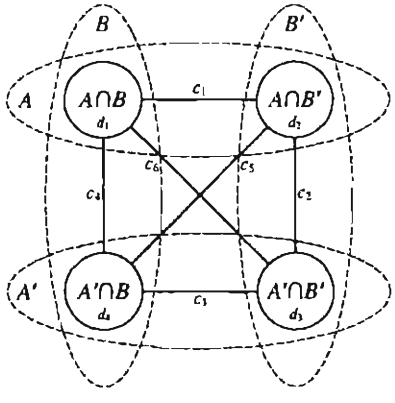


Fig. 1. Illustration to the proof of Theorem 5

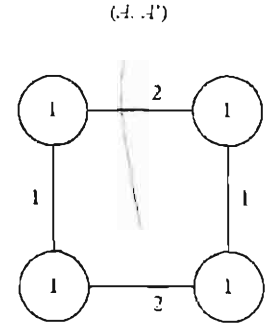


Fig. 2. A graph achieving the bound of Theorem 5

Corollary 1. *Each strict local minimum of (2, 3, 4) is not grater than two times the minimum ratio cut.*

The proof is straightforward from Theorem 3 and Theorem 5.

The bound of Theorem 5 is tight; the graph with 4 nodes shown in Fig. 2 achieves this bound. One can make larger examples easily by substituting any connected graph with sufficiently high edge capacities in place of the nodes of the given graph.

4 The Algorithm

In this section we present a heuristic algorithm based on standard nondifferentiable optimization methods for finding a local minimum of (2, 3). Finding a minimum of a nondifferentiable function is one of well- explored nonlinear programming topics [5, 18]. One of the possibilities is to approximate the nondifferentiable function with a smooth one and apply one of the well-known algorithms to find its local minimum [5, 3]. Often a better approach is to handle it directly. Indeed, in our case we obtain a very simple and fast algorithm.

Most of the optimization theory deals with convex problems for which algorithms with proven convergence can be developed. Many of these methods also work for non-convex functions finding a local minimum. However, then the convergence cannot be shown or can be shown only in a local neighborhood of some local minimum what is not satisfactory in our case. The very basic algorithm of nondifferentiable optimization is a *subgradient* algorithm [5, 18]. We will adopt it for solving our problem. Since we apply it to a non-convex problem, it should be considered mostly as a heuristic, however practice shows that it actually converges to a local minimum of our problem.

The algorithm is iterative one. The iteration of the algorithm consists of going in the negative direction of a subgradient of the function by a fixed step and then performing a projection onto the constraint (3). The constraint (4) was introduced for technical reasons required in proofs and may be not considered in the algorithm. An appropriate subgradient q of F can be calculated with the following equation for its components

$$q_i = \sum_{j \in V} \operatorname{sign}(x_i - x_j),$$

Choosing the right step size is crucial for the convergence speed. Our heuristic observation is that it should be proportional to the node spread. We choose the step equal to $\texttt{stepFactor} \cdot (x_{\max} - x_{\min})$, where `stepFactor` is some parameter. Initially $\texttt{stepFactor} = 1/14$. Its update during the algorithm is be discussed later. The projection is performed by going in the direction of a subgradient of the constraint till the constraint is reached. For the constraint H we can write its subgradient r in a different form to allow its faster evaluation

$$r_i = d_i \sum_{j \in V} d_j \operatorname{sign}(x_i - x_j) = d_i \left(\sum_{j: x_j < x_i} d_j - \sum_{j: x_j > x_i} d_j \right).$$

If we sort the x_i values and consider them in increasing order, then the needed sums can be updated incrementally leading to linear time evaluation (not counting the sorting). To perform the projection we need to calculate the step length towards the constraint. To simplify the calculations we will assume that the ordering of x_i will not change during the projection. Then the constraint function H becomes linear and the desired step can be easily calculated from the linear equation defined as the point of value 1 on the ray defined by the subgradient from the current point. In the case of unit node weights our assumption is fulfilled. To see this we must observe that the step in the function will always lead to a point with $H < 1$. Then, if we have unit node weights, $r_i \leq r_k$ for all i and k satisfying $x_i \leq x_k$, so after the projection the distance between them can only increase thus keeping the same ordering. For the case of arbitrary node weights such algorithm gives a usable approximation of the projection step length.

The whole algorithm starts with a random initialization. We assign a random position for each node such that $H < 1$ and then perform a projection to obtain a feasible starting position. We experimented also with several other initializations, but obtained significant improvements only for tree graphs. Since the optimal cut for trees can be found in linear time, we can construct a starting position that reveals it and the algorithm will only perform a few iterations to confirm that the solution does not improve. Hence to get a comprehensive picture of the algorithm behavior we decided to consider random initialization only.

After the initialization we perform iterations until convergence. To tell when the process is converged we track the minimum positional cut values obtained at each iteration. The same values will also tell us how to change the step size parameter `stepFactor`. If the new cut value is lower than the previous then we are making progress and we should continue with the same step size. If the value is higher than the previous then the step size is too large. If the cut does not change then we have a clue that the process has converged. Of course we do not hurry to make decisions only from one iteration. Instead we wait for a certain number of iterations controlled by the constants `MAX_OSCILLATIONS` and `MAX_EQUAL` before making the decision. Such delay also improves the convergence speed by allowing to iterate longer with a larger step size

To determine the positional cut value at each iteration proceed as follows. We consider the sorted sequence of nodes, calculate the positional cut in each interval between two consecutive nodes and take the minimum one. We can do it incrementally on the sorted sequence in time $O(n + m)$ provided the sorting, where m is the number of edges in the graph.

As suggested in one of the exercises in [5] the performance of this algorithm can be improved by taking the previous directions into account. We add the previous direction to the current reduced be some constant `REDUCTION_FACTOR` between 0 and 1. It models a heavy ball motion in the presence of a force in the direction of the subgradient. In our experiments such modification with `REDUCTION_FACTOR` = 0.95 performed substantially better.

All the steps described before can be implemented to run in time $O(n + m)$. Adding the time needed for sorting the nodes one iteration takes time $O(m$

+ $n \log n$). The number of iterations is hard to estimate so we will provide experimental data in the next sections. The constants MAX_OSCILLATIONS and MAX_EQUAL have the most impact on the iteration count and also on the quality of the obtained cut. So we must select them carefully. After some experimentation we chose MAX_EQUAL = 200 and MAX_OSCILLATIONS = 30.

```
algorithm ratio-cut
  calculate a random feasible initial position
  acum = 0,oscillationCounter = 0,equalityCounter = 0,stepFactor = 1/14
  while (equalityCounter < MAX EQUAL)
    x = x + acum
    d = direction(x)
    x = x + d
    acum = acum * REDUCTION FACTOR + d
    if(minimum positional cut value has increased in this iteration)
      equalityCounter = 0
      oscillationCounter ++
    else if (minimum positional cut value has decreased in this iteration)
      equalityCounter = 0
    else equalityCounter ++
    if (oscillationCounter > MAX OSCILLATIONS)
      stepFactor /= 1.3
      oscillationCounter = 0
  endwhile
end
function direction (x)
  d = subgradient(x)
  step = (xmax - xmin) * stepFactor
  x1 = x + d*step
  x2 = projection of x1 on the constraint
  return x2-x
end
```

5 The Data

We evaluated the proposed algorithm on three families of graphs: random cubic graphs, random geometric graphs and random trees. We considered only graphs with unit weight nodes and edges.

Random cubic graphs are potentially hard for ratio-cut algorithms since in [13] it was shown that there is actually a O(log n) gap between the minimum ratio-cut and the maximum concurrent flow on these graphs. We generate them using the algorithm provided in [19].

Random geometric graphs are standard test suite for balanced cut problems used in several papers [11, 25]. To generate a geometric graph we place the nodes of the graph randomly in the unit square. Then we include an edge between each two nodes that are within distance δ in the graph, where δ is the minimum value such that the resulting graph is connected.

Tree graphs are seemingly easy graphs because their optimal ratio-cut can be calculated in linear time. Also it is not hard to show that only one local minimum exists for the corresponding optimization problem. Nevertheless, it is an interesting family since our experiments indicate a slow convergence of our algorithm on these graphs. Also we can compare our result with the optimal one. We generate random trees with the classical algorithm where each tree is produced with the same probability. This algorithm produces long and skinny trees, which are particularly difficult for our algorithm.

6 Experimental Results

We implemented the algorithm in C++ and evaluated its performance on a computer equipped with a Pentium III 800 MHz processor and 256 Mbytes of RAM. For each graph family we measured the running time in seconds, the number of iterations and the quality of the produced cuts. Since we did not know the exact cut values for random and geometric graphs, we evaluated how much the ratio-cut value decreases when we continue the algorithm for the same number of iterations as performed before termination. Measuring the decrease of the ratio-cut value we can estimate how far the result is from the optimal.

The algorithm was run on series of graphs of exponentially increasing size from 100 up to 204800 nodes. Ten graphs were generated for each size and the results were averaged. The average node degree for all graph families is constant. Although we cannot specify the degree explicitly for geometric graphs, due to their nature it was about 10 on all instances. The experimental results are given in figures 3 – 8. For each graph family tables show the running time, and how much the the the ratio-cut value improves when the iteration count is doubled. Let us discuss the results separately for each graph family.

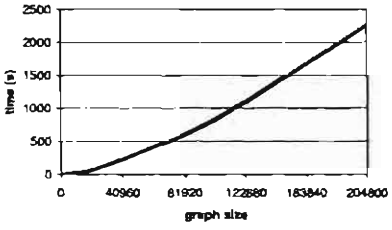


Fig. 3. Running time for cubic graphs

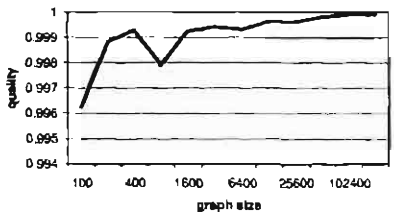


Fig. 4. Quality for cubic graphs

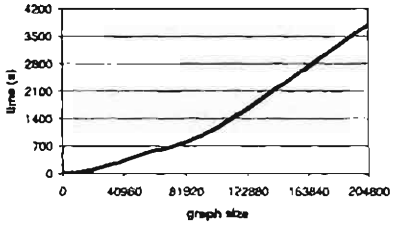


Fig. 5. Running time for geometric graphs

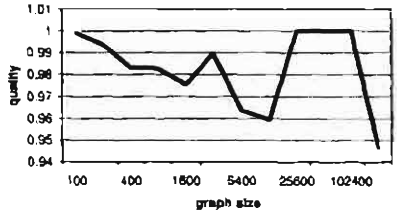


Fig. 6. Quality for geometric graphs

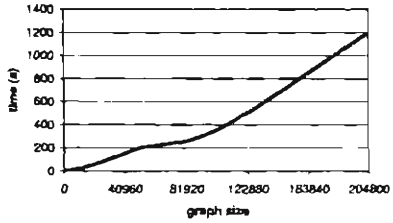


Fig. 7. Running time for tree graphs

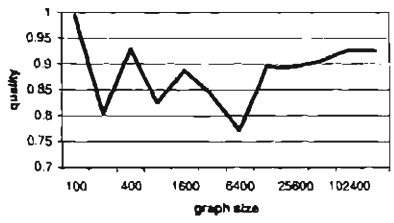


Fig. 8. Quality for tree graphs

6.1 Cubic Graphs

Although these graphs were suggested as hard, the algorithm performed very well on them. It took on average about 35 minutes to partition the 204800 node graphs. The running time dependence from the graph size is shown in Fig. 3. When we approximated the running time with a function in a form $O(n^p)$ we get the asymptotical running time about $O(n^{1.6})$ on these graphs.

The algorithm seems to find a very close to optimal cut since after doubling the iteration count the quality increased only by less than 0.4%. Even more, the quality improved for the larger graphs approaching 1 (see Fig. 4). Such behavior is not surprising since it is not hard to prove that the ratio of a cut (A, A') of a random cubic graph is $\frac{3}{n-1}$ on average independently of the sizes of A and A' (a similar proof for general random graphs is presented in [23]). Then it is unlikely that the minimum ratio cut will be much different from this average value.

6.2 Geometric Graphs

The algorithm performed very well on these graphs both in terms of speed and quality. It took on average about 1 hour to partition the 204800 node graphs. The running time dependence from the graph size is shown in Fig. 5. The asymptotical running time behavior on these graphs was about $O(n^{1.64})$. After doubling the iteration count the quality increased by less than 5% (see Fig. 6). Also visually the cuts seemed the best possible. Fig. 9 shows a typical cut of a 10000-node graph.

6.3 Tree Graphs

The running time for trees was better than for other families. The largest graphs were partitioned in about 20 minutes. The running time dependence from the graph size is shown in Fig. 7. The asymptotical running time on these graphs was about $O(n^{1.45})$. However the quality was poor. As shown in Fig. 10 the obtained cuts were far from the optimal and the quality decreased with increasing graph size. Also doubling the iteration count showed 10% to 20% quality improvement (see Fig. 8).

When we explored further the reason of the poor behavior we found out that convergence is much slower than for other graph families and the stopping

criterion does not work correctly in this case. When we allowed the algorithm to run for a sufficiently long time, it always found the optimum solution. However we did not find a robust stopping criterion that correctly works with tree graphs and does not increase running time much for other graph families. As already mentioned a smarter initialization can be used to improve the quality of the partition if such tree or tree-like graphs are common for some application.

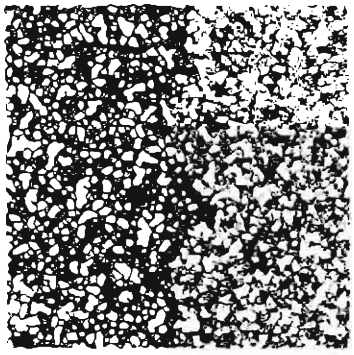


Fig. 9. The obtained cut for a 1000-node geometric graph

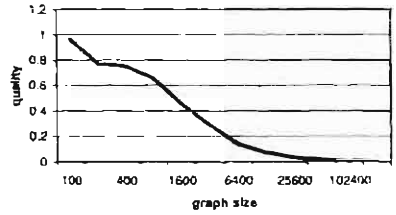


Fig. 10. Optimality for tree graphs

7 Conclusions and Open Problems

We have proposed a nondifferentiable optimization based method for solving the ratio-cut problem and presented a heuristic algorithm implementing it. We have shown that any strict local minimum is 2-optimal. The presented algorithm, however, in certain cases can find a non-strict minimum, but we can easily transform the obtained x vector into the characteristic form. Then the algorithm can be run again from this starting position and this process can be iterated until the result does not change giving a locally minimal cut, which by Theorem 5 is 2-optimal.

The obtained algorithm is simple and fast and uses amount of memory that is proportional to the size of the graph. Its running time and quality are verified experimentally. Its practical running time is about $O(n^{1.6})$ on our test data. The algorithm produce high quality cuts on random cubic and geometric graphs. On trees and other very sparse graphs the quality can be significantly improved by choosing a better starting position than a random one. We evaluated the algorithm on artificially generated data. As a further work it would be important to evaluate its performance on real-life problems. Although the algorithm performed well on most graphs, anyway it is heuristic. It is an open question whether we can find a local minimum of (2, 3, 4) in polynomial time?

8 Acknowledgements

The author would like to thank Paulis Kikusts and Krists Boitmanis for valuable discussion and help in preparation of the paper.

References

1. C. J. Alpert and A. Kahng, Recent directions in netlist partitioning: a survey, tech. report, Computer Science Department, University of California at Los Angeles, 1995.
2. C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. Proc. Design Automation Conf, pp. 530 – 533, 1997.
3. R. Baldick, A. B. Kahng, A. Kennings and I. L. Markov, Function Smoothing with Applications to VLSI Layout, Proc. Asia and South Pacific Design Automation Conf., Jan. 1999.
4. J. W. Berry and M. K. Goldberg, Path Optimization for Graph Partitioning Problems. Technical report TR: 95-34, DIMACS, 1995.
5. D. Bertsekas, Nonlinear Programming, Athena Scientific, 1999.
6. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. Handbook of Combinatorial Optimization, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.
7. K. Freivalds. A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning. Manuscript 2003. Available at *http://www.gradetools.com/karlisf/papers.html.*
8. L. Hagen and A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1074 – 1085.
9. T. Hamada, C. K. Cheng, P. M. Chau. An efficient multilevel placement technique using hierarchical partitioning. IEEE Trans. on Circuits and Systems, vol. 39(6) pp. 432 – 439, June 1992.
10. P. Klein, S. Plotkin, C. Stein, E. Tardos, Faster approximation algorithms for unit capacity concurrent flow problems with applications to routing and sparsest cuts, SIAM J. Computing, 3(23) (1994), pp. 466 – 488.
11. K. Lang and S. Rao. Finding Near-Optimal Cuts: An Empirical Evaluation. 4th. ACM-SIAM Symposium on Discrete Algorithms, pp. 212 – 221, 1993.
12. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, S. Tragoudas, Fast approximation algorithms for multicommodity flow problems. Journal of Computer and System Sciences, 50(2), pp. 228 – 243, April 1995.
13. T. Leighton, S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM, vol 46 , No. 6 (Nov. 1999), pp. 787 – 832.
14. T. Lengauer. Combinatorial Algorithms for Integrated Circuit Layout. Stuttgart, John Wiley & Sons 1994.
15. D. W. Matula, F. Shahrokhi, Sparsest Cuts and Bottlenecks in Graphs. Journal of Disc. Applied Math., vol. 27 (1990), pp. 113 – 123.
16. F. Shahrokhi, D. W. Matula, On Solving Large Maximum Concurrent Flow Problems, Proceedings of ACM 1987 National Conference, pp. 205 – 209.
17. F. Shahroki, D. W. Matula, The maximum concurrent flow problem. Journal of the ACM, vol. 37, pp. 318 – 334, 1990.
18. N. Z. Shor, Methods of minimization of nondifferentiable functions and their applications. Kiev, "Naukova Dumka" 1979. (in Russian).
19. A. Steger and N. C. Wormald, Generating random regular graphs quickly. Combinatorics, Probab. and Comput. vol. 8 (1999), pp. 377 – 396.
20. M. Wang, S. K. Lim, J. Cong, M. Sarrafzadeh. Multi-way partitioning using bipartition heuristics. Proc. Asia and South Pacific Design Automation Conf., pp. 667 – 672, 2000.
21. Y. C. Wei, C. K. Cheng, An Improved Two-way Partitioning Algorithm with Stable Performance, IEEE Trans. on Computer-Aided Design, 1990, pp. 1502 – 1511.

22. Y. C. Wei, C. K. Cheng, A two-level two-way partitioning algorithm, Proc. Int'l. Conf. Computer- Aided Design, pp. 516 – 519, 1990.
23. Y. C. Wei, C. K. Cheng, Ratio Cut Partitioning for Hierarchical Designs. IEEE Trans. on Computer- Aided Design, vol. 10, pp. 911– 921, July 1991.
24. C. W. Yeh, C. K. Cheng, and T. T. Y. Lin. A probabilistic multicommodity-flow solution to circuit clustering problems. IEEE International Conference on Computer-Aided Design, pp. 428 – 431, 1992.
25. C. W. Yeh, C. K. Cheng. T. T Y. Lin, An Experimental Evaluation of Partitioning Algorithms, IEEE International ASIC Conference, P14-1.1 – P14-1.4. 1991.

Curved Edge Routing

Kārlis Freivalds

University of Latvia
Institute of Mathematics and Computer Science
Raina blvd. 29, LV-1459, Riga, Latvia
karlisf@mii.lu.lv
Fax: +371 7820153

Abstract. We consider the problem of drawing a graph where edges are represented by smooth curves between the associated nodes. Previously curved edges were drawn as splines defined by carefully calculated control points. We present a completely different approach where finding an edge is reduced to solving a differential equation. This approach allows to represent the graph drawing aesthetics directly, even the most complex ones denoting the dependencies among the paths.

Keywords. Graph drawing, edge placement, curved edges.

1 Introduction

Edge placement is one of the fundamental problems in graph drawing [2]. There are two traditional strategies for edge placement: edges are placed together with nodes, or nodes are laid out first and then edges are routed. Placing edges together with nodes allows to create drawings with proven quality [4, 9]. However, if a more complex line style than a straight-line segment is used, it gets very hard to describe the edge influence on the node placement. The independent edge placement [6, 13] has the advantage of exploiting complex edge routing using general-purpose node layout algorithms. This is also the only strategy in interactive graph layout where the user specifies the node positions.

In this paper we follow the second strategy and assume that nodes are already positioned by the user or some layout algorithm and we have to route the edges.

There are several graphical representations of a graph [1, 2]. Nodes are basically represented by two-dimensional symbols, most commonly by upright rectangular boxes or circles. In this paper we will use only rectangular boxes. We also assume that nodes do not overlap, so there is some space for edge routing.

The visual appearance of the drawing is mostly influenced by the edge representation. Well-known edge drawing standards are orthogonal, straight-line, polyline and curved [1, 2]. Here we focus on curved edges, where the edge is represented by a smooth curve between the associated nodes.

The common approach for curved edge routing is finding a piecewise linear path which does not cross node boxes and then smoothing it using a spline. Such

approach is natural in drawing layered graphs. In layered drawings edges are replaced with polyline paths having bendpoints on every level. This construction provides a useful framework for smoothing the polyline paths with splines.

Such approach is used in the graph drawing program dot [8] where virtual node boxes are used to describe the free space in the hierarchical drawing where the path can go. At first the calculated polyline path is smoothed with a spline. Then if the spline goes outside the virtual node boxes, control points are adjusted or the spline is subdivided. In VCG [14] Bezier spline is fitted using adaptive subdivision, adjusting control points if the bounding triangle of a Bezier spline segment intersects a node or a path.

Dobkin [6] describes a more general method applicable to any graph. Again the polyline path is calculated first. Then a smooth spline is fitted, which does not intersect any obstacle. This method produces nice drawings, but only for each path separately. Crossings of several paths are not taken into account.

Obtaining a smooth path among obstacles is a well-studied problem in robot motion planning. Finding a shortest constrained curvature path of a robot in the workspace with obstacles is explored in [3]. It is shown that such path can be constructed from straight-line segments and circle arcs with a constant radius corresponding to the maximal allowed curvature.

Latombe [10] suggests a potential field method for path planning. He simulates the movement of a particle in the electrostatic field when the particle is attracted to the target and repelled from the obstacles. This seems very interesting approach, but it deals with such physical properties as speed and acceleration and application of this method to graph drawing does not seem to be straightforward.

Sethian [15] develops a general theory called level set methods and shows how to apply it to produce a shortest path of a robot within the environment of constraints. He reduces the problem to a differential equation and applies the fast marching method to solve it. We have found another application of this theory. Using Sethian's approach it is easy to find curves according to specific rules. In our case the rules are constructed in a special way to reflect the aesthetics of graph drawing and allows finding the path directly, avoiding usage of splines.

The rest of the paper is organized as follows. In Section 2 we define the aesthetic edge routing problem by choosing a cost function and then finding the edge drawing according to this cost function. Section 3 gives details how to construct the cost function to represent the curved edge drawing aesthetics. Section 4 presents an algorithm to route a path according to the given cost function. In Section 5 we give the analysis of the results and sketch the possible improvements.

2 Problem definition

At first we have to find out how the well-routed edges should look like. There are a number of commonly accepted aesthetics criteria for graph drawing [6, 2, 9]. We consider the following for curved edge drawings:

(1) edges are short,
(2) edges are well separated from node boxes and other edges,
(3) edges do not turn sharply,
(4) edge crossing number is small,
(5) edges cross in wide angles.

An intuitive way how to represent these aesthetics is to assign some cost $h(x, y)$ to every point (x, y) on the plane expressing how good it is for the edge to go through this point. Then the edge is searched as a path between its node boxes that has the smallest total cost. For now we can assume that the path is joining the node centers (x_0, y_0) and (x_1, y_1). Formally each individual path can be expressed as a curve L with fixed endpoints (x_0, y_0) and (x_1, y_1) which minimizes the integral

$$\int_L h(x, y)dL. \tag{1}$$

This definition corresponds only to one path, but the aesthetics criteria express also relations between several edges. To overcome this limitation we construct the paths one by one but incorporating the previously routed paths into the cost function.

At the beginning we have only nodes, so the cost function is calculated for them. When a new path is routed the cost function is updated to reflect the current configuration. Since we are adding paths one by one the drawing is dependent on the order in which they are added. To reduce the sensitivity to the order, paths are routed iteratively e.g. after initial routing they are rerouted several times by taking one path at a time, deleting it and routing it again. This method is summarized in the algorithm *routeEdges*.

```
algorithm routeEdges
  Build the initial cost function corresponding to nodes
  for each path P
    Route P
    Update the cost function with P
  endfor
  for iter = 1 to MAX_ITER
    for each path P
      Delete P from the cost function
      Route P
      Update the cost function with P
    endfor
  endfor
end
```

Two parts of the algorithm need to be explained: how to build the cost function, and how to find the path corresponding to the defined cost function. These problems are discussed in the following sections.

3 Finding the cost function

Let us look how to construct the cost function $h(x,y)$ to incorporate the above mentioned aesthetics. We express $h(x,y)$ as a weighted sum of functions, where each function expresses the influence of one drawing element (node or path). We add also some constant c_1, which expresses the cost of the length of the path. The general form of the function is

$$h(x,y) = c_1 + c_2 \sum_i b_i(x,y) + c_3 \sum_i p_i(x,y), \tag{2}$$

where b_i are the functions for nodes and p_i are the functions for paths. For nodes we want the function to be infinite inside the nodes to prohibit the paths from crossing them, and decreasing with the distance from nodes to keep the paths some distance apart from them. For paths we want the function to be of some finite magnitude near the path and decreasing with the distance from the path. Constants c_2 and c_3 are weighting factors for the influence of the nodes and paths respectively. We found that setting $c_1 = 1$, $c_2 = 1.5$ and $c_3 = 5$ is adequate.

Denoting the distance from a point (x,y) to the drawing element by $d(x,y)$ our choice for the function for nodes was

$$b(x,y) = \begin{cases} \frac{1}{d(x,y)} & \text{if } (x,y) \text{ is outside the node} \\ \infty & \text{if } (x,y) \text{ is inside the node} \end{cases} \tag{3}$$

In practice the value of b_i inside the node is set to a large enough constant for paths not to cross the node. This simplifies the algorithm and allows us to route the paths between the node centers.

For paths two kinds of functions were considered:

$$p(x,y) = \max(1 - \frac{d(x,y)}{\sigma}, 0), \tag{4}$$

$$p(x,y) = e^{-\frac{d(x,y)^2}{\sigma^2}}, \tag{5}$$

where σ controls the desirable separation between paths. In the Fig. 1 we can see the drawings of a graph using different cost functions for the paths. Both functions give good results, but the second one overall seemed to be a little better, so further we will work with it. For the further discussion we must choose a good representation of the cost function. We chose the simplest approach to represent the cost function by sampling values on the orthogonal grid. Other choices are also possible, for example triangular meshes, but that seemed to be more complex.

Let us look how to calculate the cost function on the grid quickly. We have to notice that the functions b_i and p_i have some influence only in a small neighborhood near the corresponding object. In other points we can set the function to zero without affecting the resulting drawing too much. We need to find all the points where the function is large enough.

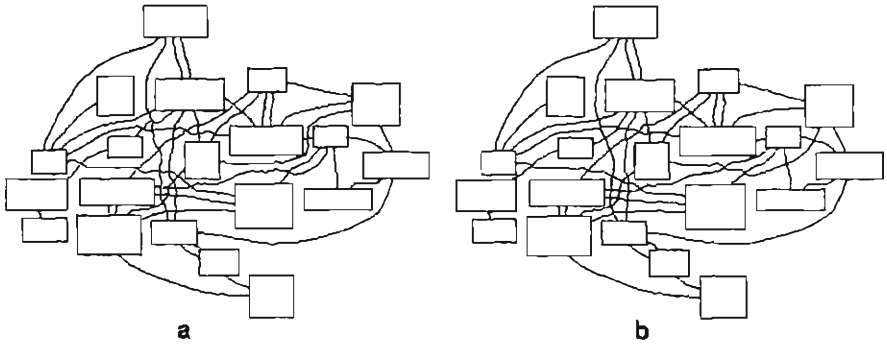


Fig. 1. Drawings of a graph using linear cost function for paths (a), and exponential one (b)

For node boxes this is easy since all such points are contained in a rectangle around the node. Size of this rectangle can be easily calculated from the definition of b_i.

For paths the situation is more complex, because the path is approximated with many short line segments and determination of the distance for the point to the path requires testing all the path segments. We use the property that our functions are monotonically decreasing with the distance from the object.

The algorithm *cost_function_for_path* calculates the function by starting at the path and expanding the zone if the function is greater than some threshold. For this purpose a priority queue ordered by the function p value is used. For each point in the queue the nearest path segment is kept and the function is calculated depending only from this one segment. The queue is initialized with the midpoints of all the segments. In the loop of the algorithm we get the point with the highest function value from the queue and calculate the function for its neighbors. If the function is larger than the given threshold for them, we put them into the queue. Since the priority queue can be implemented that one operation takes $O(\log n)$ time [5], it can be easily seen that this algorithm finds the correct function in time $O(n \log n)$, where n is the number of the affected points.

```
algorithm cost_function_for_path
  for each segment S of the path
    M = midpoint of S
    d = distance from M to S
    Calculate the function value h depending on the distance d
    Put < M,S,h > into the priority queue
  endfor
  while the queue is not empty
    Get < M,S,h > with the largest h value from the queue
    if h < threshold then terminate
    for each of the four neighbour points N of M
```

```
      if the function has not been calculated for N then
         d = distance from N to S
         Calculate the function value h1 depending on d
         Put < N, S, h1 > into the priority queue
      endif
    endfor
  endwhile
end
```

4 Finding the path

For the given cost function our task is to find the path between two given node centers. The simplest approach would be to search for the shortest path in the grid neighbor graph. But this does not give acceptable solution because of the orthogonal structure of the graph. We need a continuous, direction independent solution approximated on the grid.

We use a method essentially the same as the Dijkstra's shortest path searching [5, 11]. At first the distance map is calculated for all points and then the path is determined by tracing backwards from the destination point. But there are differences in the distance map calculation and back tracing. In the Fig. 2 we can see the cost function and the distance map of the cost function shown in Fig. 5. The starting point was chosen in the middle of the map.

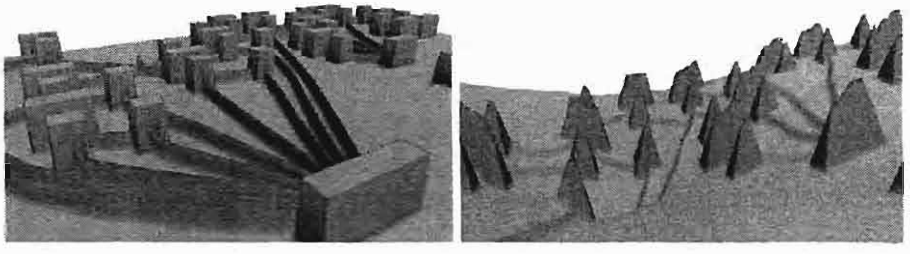


Fig. 2. The cost function (*left*) and the corresponding distance map (*right*) plotted in three dimensions

4.1 Calculation of the distance map

The distance map is a function $f(x, y)$ which denotes the distance from the starting point (x_0, y_0) in the environment of varying costs. It is easy to imagine the distance map as a pit-like function in three dimensions, where the height of the pit surface is the value of the distance map at the given point. It is very interesting that this pit is the same as develops when the sand of non-uniform structure run out from the box with a little hole at the bottom. If the sand can

hold at the maximum slope $h(x,y)$ then it is exactly our distance map. Moreover this construction has a prototype in the mining industry when an open pit must satisfy the given slope requirements [12]. The distance map $f(x,y)$ satisfies the following equation

$$||\nabla f|| = h(x,y). \tag{6}$$

This means that the slope of f is our cost function h. After adding the starting condition and writing the gradient explicitly the problem can be stated as follows:

$$\begin{cases} \left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 = h(x,y)^2 \\ f(x_0,y_0) = 0 \end{cases}. \tag{7}$$

This equation can be solved with numerical methods by approximating derivatives with differences. For example the central differences can be used:

$$\begin{aligned} D_x &= \frac{f(x+1,y)-f(x-1,y)}{2}, \\ D_y &= \frac{f(x,y+1)-f(x,y-1)}{2}, \end{aligned} \tag{8}$$

and the problem is transformed to

$$\begin{cases} D_x^2 + D_y^2 = h(x,y)^2 \\ f(x_0,y_0) = 0 \end{cases}. \tag{9}$$

The solution can be obtained by iterative solving the equation at each grid point. But this is very slow because many iterations are needed.

Sethian [15] proposes a faster method. He observes that the solution can be constructed by propagating the front like in Dijkstra's shortest path algorithm. He uses one-side differences instead of central ones selecting the direction, which the front came from. As a result one pass over the grid is sufficient. According to Sethian the algorithm works as follows:

```
algorithm distanceMap
  for all (x,y) f(x,y) = ∞
  f(x0,y0) = 0
  Put (x0,y0) into the priority queue
  while the queue is not empty
    Get and remove the point (x,y) with the smallest f value
    from the queue
    if (x,y) = (x1,y1) then terminate
    for each of the four neighbour points (u,v) of (x,y)
      if f(u,v) = ∞ then
         compute value of f(u,v) by selecting the largest solution
         z of the quadratic equation:
           (max(z − min(f(u − 1,v), f(u + 1,v)), 0))^2 +
           (max(z − min(f(u,v − 1), f(u,v + 1)), 0))^2 = h(u,v)^2
         Put (u,v) into the priority queue.
      endif
```

```
    endfor
  endwhile
end
```

The algorithm terminates when the priority queue is empty or the destination point is reached. In [15] Sethian proves the correctness of the algorithm as well as the time complexity of $O(n \log n)$, where n is the number of affected grid points.

4.2 Back tracing

Now when we have the distance map calculated we can search for the path. The simplest way is to trace back from the current point to its lowest neighbor, starting at the other endpoint of the path. This approach does not find the best path because of the two dominant directions imposed by the grid.

Another way is to walk in the direction of the gradient with a fixed step. This approach is rather complicated because a sub-cell accuracy is needed.

We have implemented a different approach (algorithm *findPath*) similar to the error diffusion paradigm in computer graphics [7]. We walk gridpoint by gridpoint like in the first approach, but correcting the direction pointed by the gradient. At the current gridpoint we calculate the gradient, normalize it and walk to the neighboring gridpoint, which is the closest to the gradient. Then the error is calculated by subtracting the gradient vector from the direction moved. This error is added to the gradient for the next point.

```
algorithm findPath
  error = 0
  P = (x1, y1)
  while P ≠ (x0, y0)
    add P to the path
    calculate the gradient G at the point P
    G = G / |G|
    G = G + error
    V = P + G
    P = the closest gridpoint to V
    error = V − P
  endwhile
end
```

The walk always terminates in the starting point because from its construction $f(x, y)$ contains only one minima at the starting point. This method gives us the exact curve approximated on the grid as a chain of gridpoints. In the Fig. 3 we can see the paths generated by the lowest neighbor walk and the algorithm findPath using unity cost function. It can be seen that the latter produces a perfectly straight line on the grid. Fig. 4 shows the path corresponding to a realistic cost function.

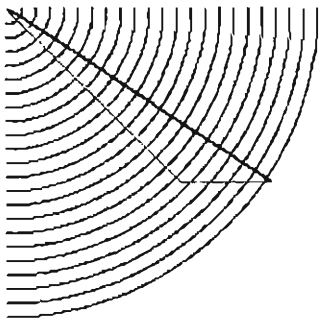


Fig. 3. Paths generated by the lowest neighbor walk (*thin*) and the algorithm findPath (*thick*) using the unity cost function

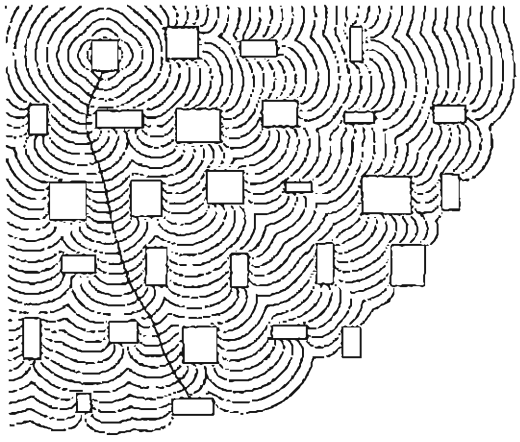


Fig. 4. A distance map and a path corresponding to it

To reduce the memory required for the path representation the path is approximated by a polyline. A simple algorithm is used where the sequence of gridpoints is approximated by a line segment if all points approximately fall on a common line. This algorithm has a quadratic worst-case complexity but in practice it is very fast in comparison with other parts of the program.

5 Implementation and results

We have implemented the curved path router as a separate program, which takes graph description as an input and routes all the paths. The input must contain node positions and sizes and the edge connectivity information.

In Fig. 5 to Fig. 8 we can see some drawings produced by the proposed edge placement algorithm. The drawings of the edges appear natural, easily perceptible for the user. The drawing conforms to all the mentioned aesthetics. The relations between several paths are represented correctly, edge separation is good and edges cross in wide angles, what is hard to achieve with previously known approaches [6, 8, 14].

The desired behavior is easily customizable by adjusting the weights for nodes or paths, or changing the cost function to incorporate new aesthetics.

Sometimes when the node is large the paths coming from node center leads to uneven distribution of the edges along the node border. To remedy this we could alternatively modify the distance map calculation by starting at all the border of the start node and ending when the border of the destination node is reached.

Our implementation of the proposed edge routing approach is not fast. For example it took 12 seconds to route the edges of the graph shown in the Fig. 5 on a Pentium III 800 MHz computer. The most expensive part is the distance

map calculation. We have observed that its time complexity of routing one edge is approximately $O(L^2 \log L)$, where L is the length of the path in grid units.

The speed of the program was not the goal of this research, so there is much room for optimizations. The grid step has the greatest impact on the running time, so it is crucial to choose the right value to compromise the quality with the running time. We chose the grid size equal to the pixel size on the screen for the best visual quality. However we believe that more sophisticated choices dependant from the characteristics of the graph are possible.

Another improvement could be propagating the front only in the most promising directions. However our first experiments using goal directed search [11] failed, because in Sethian's approach the propagation must be carried out in a strict distance ordering.

The number of iterations also plays a significant role. In the most cases 3 to 5 iterations are sufficient, so in our experiments we always make 5 iterations. Some adaptive criteria could do better. The initial ordering of the paths is also important. May be it would be worth routing the shortest paths first. Thus we might reduce the iteration count and also the crossing number could be smaller.

References

1. G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography, *Computational Geometry: Theory and Applications*, vol. 4, no. 5, 1994, pp. 235-282.
2. G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis. Graph Drawing. Prentice Hall, 1999.
3. A. Bicchi, G. Casalino, C. Santilli. Planning Shortest Bounded-Curvature Paths for a Class of Nonholonomic Vehicles among Obstacles. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp.1349-1354, 1995.
4. C. C. Cheng, C. A. Duncan, M. T. Goodrich, S. G. Koburov. Drawing Planar Graphs with Circular Arcs. Symp. on Graph Drawing GD'99, *Lecture Notes in Computer Science*, vol.1731 , pp.117-126, 1999.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to Algorithms. MIT Press. Cambridge, Massachusetts. xvii, 1990.
6. D.P. Dobkin, E.R. Gansner, E. Koutsofios, Stephen C. North. Implementing a General-Purpose Edge Router. Symp. on Graph Drawing GD'97, *Lecture Notes in Computer Science*, vol.1353 , pp.262-271, 1998.
7. J. D. Foley, A. vanDam, S. K. Feiner, J. F. Huges, Computer Graphics: principles and practice, Addison-Wesley, 1993.
8. E. R. Gansner, E. Koutsofios, Stephen C. North, Kiem-Phong Vo. A Technique for Drawing Directed Graphs. TSE 19(3), pp.214-230 (1993)
9. M. T. Goodrich, C. G. Wagner. A Framework for Drawing Planar Graphs with Curves and Polylines. Symp. on Graph Drawing GD'98, *Lecture Notes in Computer Science*, vol.1547 , pp.153-166, 1998.
10. J.C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, Boston, 1991.
11. T. Lengauer. Combinatorial Algorithms for Integrated Circuit Layout. John Wiley & Sons, 1990.

12. H. Lerchs, I. G. Grossmann. Optimum Design of Open-Pit Mines. Transactions, C.I.M., Volume LXVIII, 1965, pp.17-24.
13. K. Miriyala, S. W. Hornick, R. Tamassia. An Incremental Approach to Aesthetic Graph Layout, *Proc. International Workshop on Computer-Aided Software Engineering*, 1993.
14. G. Sander. Graph Layout through the VCG Tool. Technical Report A03-94, Universität des Saarlandes, FB 14 Informatik, 1994.
15. J.A. Sethian. Level Set Methods. Cambridge University Press, 1996.

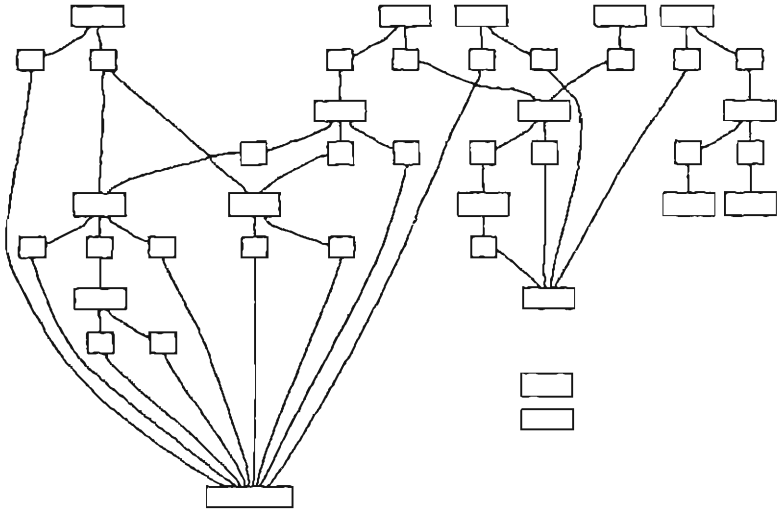


Fig. 5.

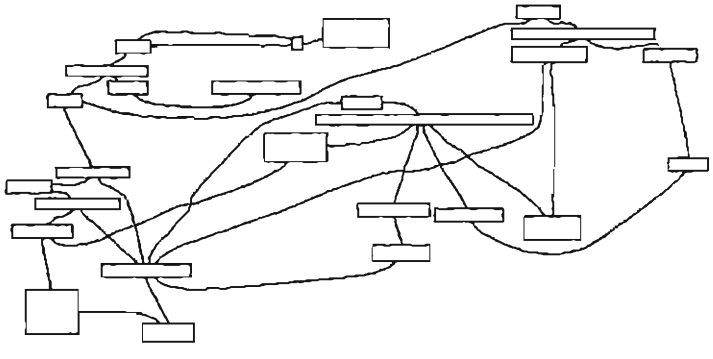


Fig. 6.

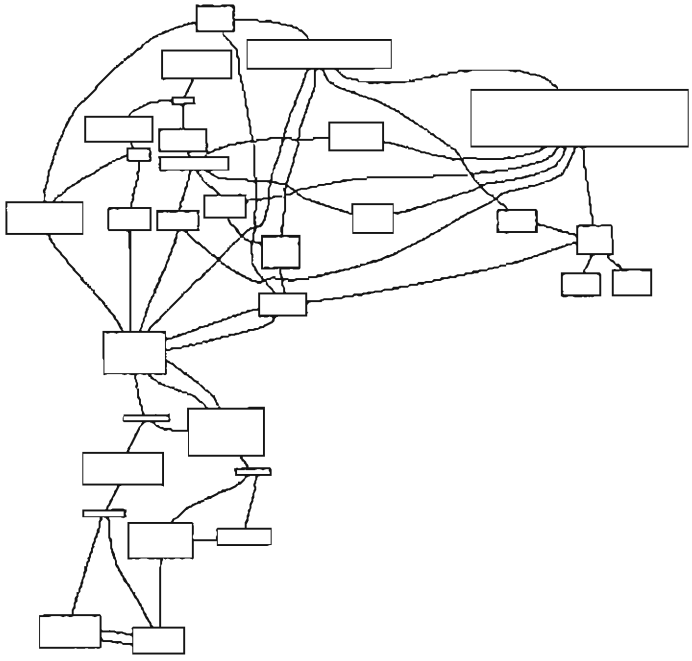


Fig. 7.

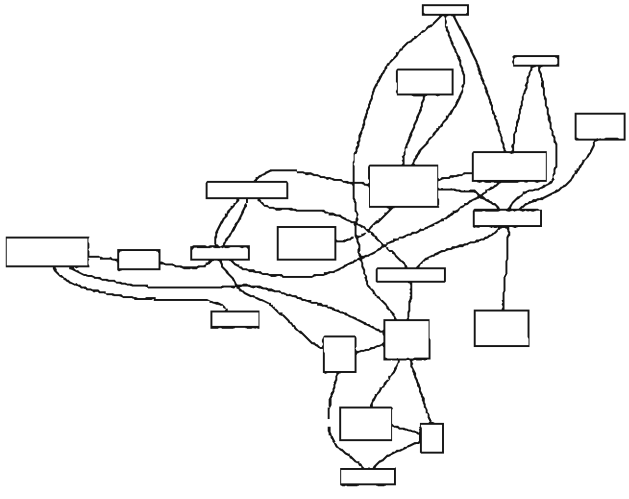


Fig. 8.

Disconnected Graph Layout and the Polyomino Packing Approach

Karlis Freivalds[1], Ugur Dogrusoz[2], and Paulis Kikusts[1] **

[1] Institute of Mathematics and Computer Science, Univ. of Latvia, Riga, Latvia
{karlisf,paulis}@mii.lu.lv
[2] Computer Engineering Department, Bilkent Univ., Ankara, Turkey
ugur@cs.bilkent.edu.tr

Abstract. We review existing algorithms and present a new approach for layout of disconnected graphs. The new approach is based on polyomino representation of components as opposed to rectangles. The parameters of our algorithm and their influence on the drawings produced as well as a variation of the algorithm for multiple pages are discussed. We also analyze our algorithm both theoretically and experimentally and compare it with the existing ones. The new approach produces much more compact and uniform drawings than previous methods.

1 Introduction

Graphs model the complex information of a system of discrete objects and their relationship. *Graph layout* is the automatic positioning of the nodes and edges of a graph in order to produce an aesthetically pleasing drawing that is easy to comprehend [5, 8].

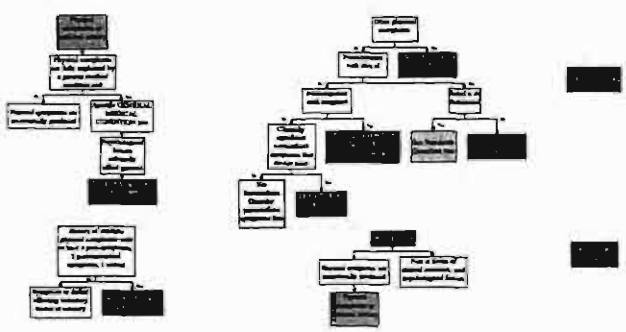


Fig. 1. An example of a disconnected graph.

** Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162 and Tom Sawyer Software, Oakland, CA, USA.

Many graph layout and editing systems have been developed in the past [5, 7, 11]. One essential aspect that has not been addressed sufficiently, is the layout of disconnected graphs; that is, the placement of the components (possibly consisting of a single isolated node) of a disconnected graph. Disconnected graphs occur rather frequently in real life applications either during the construction of a graph interactively or because of the nature of the application (Figure 1).

Most graph layout algorithms assume a graph to be connected and try to minimize the area needed for the resulting drawing. No matter how effective such an algorithm is, the space wasted overall could be arbitrarily large if the relative locations of disconnected objects of a graph are chosen by a naive, inefficient method.

Another key parameter here is the aspect ratio of the region (e.g., a window) within which the graph is to be displayed (Figure 2). When displaying a graph, the larger the wasted space is, the less visible objects will be, making the visualization process more difficult. Thus, a disconnected graph layout algorithm must strive for a packing of disconnected objects which respects the aspect ratio of the region in which it is to be displayed.

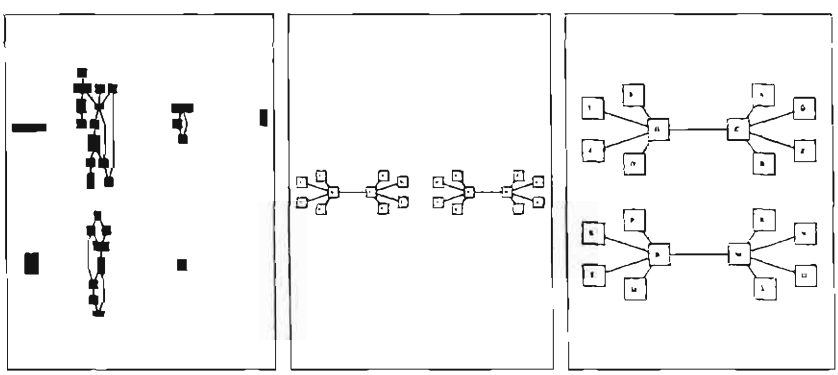


Fig. 2. How a naive disconnected graph layout algorithm can make inefficient use of the area (**left**), and why the aspect ratio of the region in which the graph is to be drawn should be taken into account during disconnected graph layout (**middle and right**).

In this paper, we review existing two-dimensional packing algorithms for the layout of disconnected graphs for a specified aspect ratio based on *strip-packing*, *tiling*, and *alternate-bisection* methodologies [6], and introduce a new algorithm that represents disconnected objects with polyominoes as opposed to rectangles. We also discuss the experimental results obtained and compare our algorithm with the previous ones. As expected, the new approach, which uses a more accurate representation for the objects, produces much more compact results.

The drawings are also more aesthetically pleasing as the new approach places the objects more uniformly.

2 Definitions and Basics

Throughout the paper, the terms "graph object", or simply "object", are used interchangeably to denote a component or an isolated node of the graph to be laid out.

The tightest rectangle bounding the drawing of a graph object or the entire graph is said to be its *bounding rectangle*. The size of a graph's drawing is identified with its bounding rectangle's. The *aspect ratio* of a rectangle $R = (W, H)$ is equal to W/H.

Most layout algorithms represent graph objects with either points or rectangles in the plane. A *polyomino* is a geometric figure formed by joining unit squares at the edges. In our new approach we use polyominoes to represent graph objects (Figure 3).

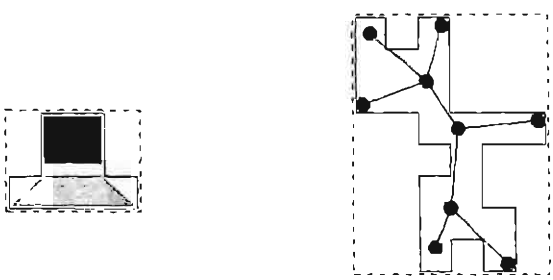


Fig. 3. Polyominoes facilitate more accurate geometric representation for graph objects. Two different representations of an isolated node and a graph component: rectangle (dashed) and polyomino (solid).

The task for a disconnected layout algorithm is to position a set of objects represented by rectangles or polyominoes with ordered dimensions (i.e., no rotations allowed) such that no pair of objects overlap and the area of the bounding rectangle of the drawing is minimized, respecting the aspect ratio of the region in which the graph is to be displayed. There has been extensive research done on two-dimensional packing of rectangles [3, 1, 4]. In the graph layout version of the problem, the user also specifies a *desired aspect ratio* for the resulting drawing so that the scaling that needs to be done before displaying the graph is minimal (Figure 2).

For an arbitrary list of n objects L_n, or simply L, let $\mathrm{A}^A(L)$ denote the area actually used by a particular algorithm A when applied to L. The *wasted space* is the unoccupied area of the packing: $\mathrm{WS}^A(L) = \mathrm{A}^A(L) - \sum_{i=1}^{n} \mathrm{A}_i$, where A_i is the area of object L_i. Similarly, the *fullness* of a packing expresses, in

percentage, how effectively the area is used by the packing algorithm: $F^A(L) = 100 \cdot (\sum_{i=1}^{n} A_i)/A^A(L)$. The *adjusted fullness* of a packing $AF^A(L)$ ($\leq F^A(L)$), expresses the fullness of a packing, in percentage, with respect to the desired aspect ratio. To be precise, it considers the additional area wasted when the final drawing is displayed in a region of desired aspect ratio DAR: $AF^A(L) = F^A(L) \cdot \frac{AR^A(L)}{DAR}$ where $AR^A(L)$ is the aspect ratio of the packing produced by algorithm A when applied to objects L and we assume $AR^A \leq DAR$. Figure 4 illustrates this with an example, in which $F^A(L) = \frac{A}{A+B}$, whereas $AF^A(L) = \frac{A}{A+B+C}$.

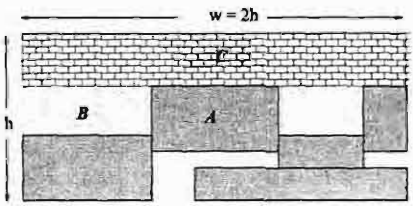


Fig. 4. Total area in which rectangles are packed is divided into three disjoint regions A (rectangles), B (wasted area), and C (additional area wasted when displayed in a region of aspect ratio 2).

Strip-Packing: One can find substantial literature on the design and analysis of algorithms for two-dimensional packing [1, 3, 4], the most popular version being *strip-packing*. In strip-packing, given a list of $n \geq 1$ rectangles $L_n = (R_1, \ldots, R_n)$, each having ordered dimensions (W_i, H_i), they are to be packed into a semi-infinite strip of unit width without any overlaps in order to minimize the height of the packing. This problem has applications in many areas including stock-cutting, two-dimensional storage problems, and resource-constrained scheduling in computer systems [2].

The most popular approach to strip-packing is the level algorithms. *First-Fit Decreasing Height* (FFDH) is a level algorithm, in which, at any point in the packing sequence, the next rectangle to be packed is placed left-justified on the first level on which it will fit. If none of the current levels will accommodate this rectangle, a new level is started. *Best-Fit Decreasing Height* (BFDH) is similar to FFDH except that the rectangles are packed, whenever possible, on current levels where they fit best.

For an arbitrary list of n rectangles L_n, all assumed to have width no greater than 1, $OPT(L)$ denotes the minimum possible bin height within which rectangles in L can be packed.

Ordered One-Dimensional Packing: For a set of rectangles, *one-dimensional packing* or simply 1D packing along x-axis (y-axis) corresponds to the process of ordering these rectangles with respect to their x-coordinates (y-coordinates) without any overlaps to *minimize* the total width (height) of the bounding rect-

angle. If the current relative positions of rectangles are to be preserved in the packing, we call it *ordered 1D packing*. Ordered 1D packing of n objects can be performed in $O(n \log n)$ time [12].

3 Related Work

In this section, we review the existing algorithms for disconnected graph layout, which represent disconnected objects with rectangles. Detailed information on these algorithms may be found in [6].

3.1 Strip-Packing Method

This method directly applies a known strip-packing algorithm such as BFDH. The width of the strip (equivalently, the factor by which the rectangle dimensions are to be scaled) is calculated based on the desired aspect ratio, using the theoretical performance of the strip-packing algorithm. With BFDH, assuming object dimensions to be independent uniform random samples from the interval $[0, 1]$ and $\mathrm{OPT}(L) \approx n/4$, the expected value of adjusted fullness, $E[\mathrm{AF}^{BFDH}(L_n)]$, is shown to be 58.8 [6]. However, these calculations are based on the worst-case performance bounds and are rarely met in practice, making it not a particularly good "guess" for the bin width.

The algorithm is of $O(n \log n)$ time complexity.

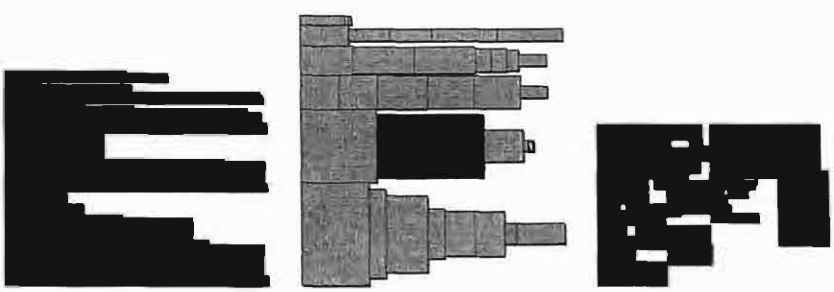


Fig. 5. The same graph laid out with strip-packing, tiling, and alternate-bisection methods, respectively for desired aspect ratio 1.0.

3.2 Tiling: Strip-Packing with Variable Width Strip

The tiling method eliminates the need to "guess" the right size strip by maintaining a bin whose width *dynamically* changes (i.e., increases). The algorithm starts by creating an initial level and placing the first rectangle in this level. It proceeds by determining whether the next rectangle in line should be added to

one of the existing levels (the one which is the least utilized at the moment) or to a newly created level. The rectangle is tiled on one of the existing levels if there is enough room. Otherwise, a decision is made on whether the current strip width should be enlarged or a new level should be formed to keep the aspect ratio closer to the desired one.

In general, the tiling algorithm does not assume any particular ordering of the objects. However, experiments show that when graph objects are sorted in nonincreasing height, most compact drawings are obtained. Notice that when objects are processed in order of nonincreasing height, the algorithm turns into a variation of a strip-packing algorithm, BFDH to be more specific, where the strip width is dynamically increased as necessary to better fulfill the aspect ratio constraint.

The algorithm is of $O(n \log n)$ time complexity.

3.3 Alternate-Bisection Method

This divide-and-conquer method works by bisecting the disconnected objects of a graph alternately as follows. The objects are bipartitioned using a metric such as total area and objects in each partition are recursively laid out. The recursion continues until a partition consists of a small, constant number of objects (e.g., one) whose optimal layout becomes easy if not trivial. At the end of each recursive step, when placing the two embedded partitions relatively, the orientation is alternated. For instance, the last step would place the two already positioned partitions side by side (horizontally) if the four partitions in the previous step were placed one on top of the other (vertically) pairwise.

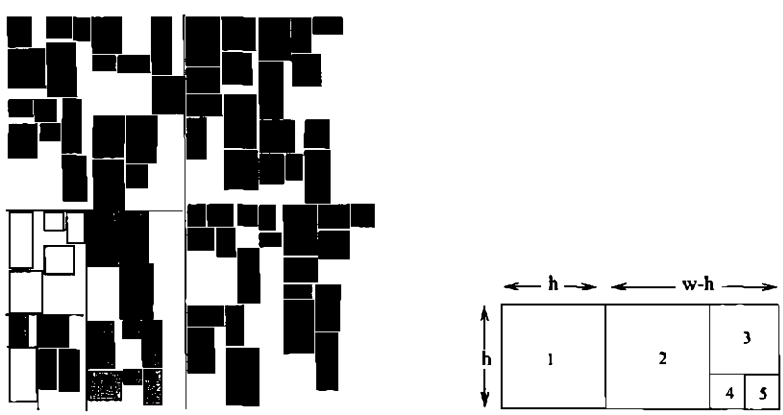


Fig. 6. An example of the alternate-bisection method; on one branch of the recursion, alternately partitioned objects are shown with separating lines and different colors **(left)**. An example of how the alternate-bisection method can be adapted to an arbitrary aspect ratio **(right)**.

The theoretical analysis prove that the total area wasted by the algorithm for n objects, $W(n)$, is roughly $O(n^{1.41})$ [6], which is quite inefficient. However, when simple alternating ordered 1D packings are applied in each recursive step (e.g., objects in upper (lower) left partition are packed downwards (upwards), towards the horizontal separating axis in Figure 6), the experimental results show that much more compact results are obtained [6]. The overall time complexity of the algorithm is $O(n \log^2 n)$.

For independent, uniformly distributed random object dimensions, this algorithm will not favor one orientation over the other and yield "square-like" drawings. The desired aspect ratio can be respected by this algorithm by initially recursively partitioning the set of objects into two, one partition to be laid out with aspect ratio 1.0 and the other with $\mathrm{DAR}(L) - 1.0$ $(= \frac{w-h}{h})$, assuming $\mathrm{DAR}(L) = \frac{w}{h} > 1.0$ (Figure 6). Alternatively, the object dimensions can be scaled with respect to the desired aspect ratio as a preprocessing step, after which, the desired aspect ratio may be assumed to be 1.0.

3.4 Comparison of the Methods

In [6], experiments with graphs laid out with random aspect ratio and with random object dimensions are presented. Notice here that the graph objects are represented with rectangles and the area wasted by such representation is ignored. In the context of graph layout, it is argued that the object dimensions are not completely of uniform distribution since the two types of disconnected objects, isolated nodes and larger components, in most cases will be of highly varying dimensions. Experiments conducted with two groups of objects with dimensions uniformly distributed within each group but with different means are presented in Figure 7.

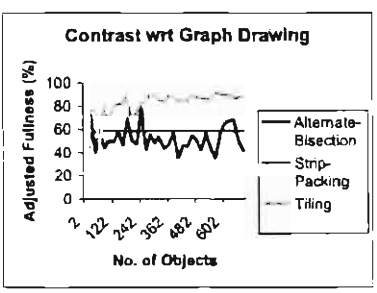


Fig. 7. Comparison of the performance of the three methods using a distribution model of dimensions that is more suitable in the context of graph layout.

In terms of execution time, both split-packing and tiling methods are superb since they are of $O(n \log n)$. The alternate-bisection method, on the other hand, gets a little slow as the number of objects are over a thousand. Considering that

most interactive graph drawing applications will not consist of more than, say one hundred, disconnected objects, this method is also of practical value.

In terms of the quality of the packings produced, the experiments show that the tiling method clearly produces the most compact drawings. However, the results obtained from the alternate-bisection method tend to be more aesthetically pleasing since the objects are generally distributed more uniformly (Figure 5).

4 Polyomino Packing Approach

In this approach each graph object is represented by a polyomino. We define a polyomino as a finite set of $k \geq 1$ cells of the infinite planar square grid G that are fully or partially covered by the drawing of the object. If the case that an object is placed completely inside another one is not desirable, the definition can be modified and the uncovered grid cells that are completely bounded by the covered ones can be included as well.

Given a set of polyominoes $P_i, 1 \leq i \leq n$, packing them into a minimum area is clearly NP-hard, even when the polyominoes are restricted to rectangles [9]. Our heuristic algorithm for polyomino packing is a greedy one: it places the objects one by one, finding the optimal place for the new object, one at a time, with respect to the already placed ones. The optimal place for a polyomino is simply calculated as the grid cell G_{xy} located at (x, y) where the function $\max(|x|, |y|)$ is minimized over all grid cells. The cost function defines the order in which the cells are examined and this order is the same for all polyominoes.

A grid data structure is used to represent free grid cells, which are later marked as occupied as polyominoes are placed. In order to find the best place the algorithm PACKPOLYOMINOES looks sequentially through all cells in the increasing order of the cost function defined above. If an available spot (a set of unoccupied grid cells where the polyomino fits) is found, it is placed there and the corresponding grid cells are marked as occupied. When testing for intersections, we simply go through all polyomino cells and test whether each can be placed in a free grid cell.

Our experiments show that the quality of the packing depends very much on the order in which the polyominoes are processed. The best results are obtained when they are ordered and processed from the largest to the smallest, which conforms to the ordering in heuristic approaches for the bin packing [2] and strip-packing problems [3]. The size of each polyomino can be defined in several ways including its number of cells (i.e., area) and the perimeter of its bounding rectangle. Experiments show that both give similar results, so we choose the perimeter of the bounding rectangle for calculating the object sizes for the ease of implementation.

Here is a pseudo code of our algorithm:

algorithm PACKPOLYOMINOES($P_i, 1 \leq i \leq n$)
(1) sort $P_i, 1 \leq i \leq n$ in the order of nonincreasing size
(2) initialize the grid G using the sizes of $P_i, 1 \leq i \leq n$
(3) **foreach** polyomino P_i **do**

(4) calculate (x, y) such that the cost function is minimized
(5) **while** cannot place P_i in G centered at (x, y) **do**
(6) calculate next (x, y) using the cost function
(7) **end while**
(8) mark the cells in G covered by P_i as occupied
(9) **end foreach**

Figure 8 illustrates an example drawing produced using our algorithm for the component placement of a forest. Also see Figure 9 for the drawing produced by our algorithm for the graph in Figure 1.

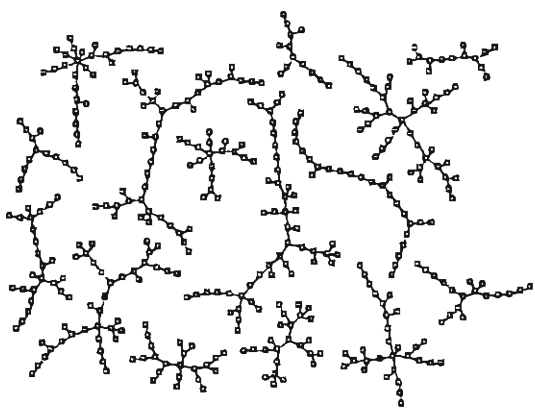


Fig. 8. An example packing produced by our algorithm.

4.1 Parameters

The grid step l is obviously the most significant parameter of this approach. We would like to guarantee that the average polyomino size s is not exceeding some constant c:

$$s = \frac{1}{n}\sum_{i=1}^{n}\lceil\frac{W_i}{l}\rceil\lceil\frac{H_i}{l}\rceil \leq c$$

$$\Rightarrow \quad \frac{1}{n}\sum_{i=1}^{n}(\frac{W_i}{l}+1)(\frac{H_i}{l}+1) \leq c$$

$$\Rightarrow \quad \sum_{i=1}^{n} W_i H_i + l\sum_{i=1}^{n}(W_i + H_i) + l^2 \leq cnl^2$$

10

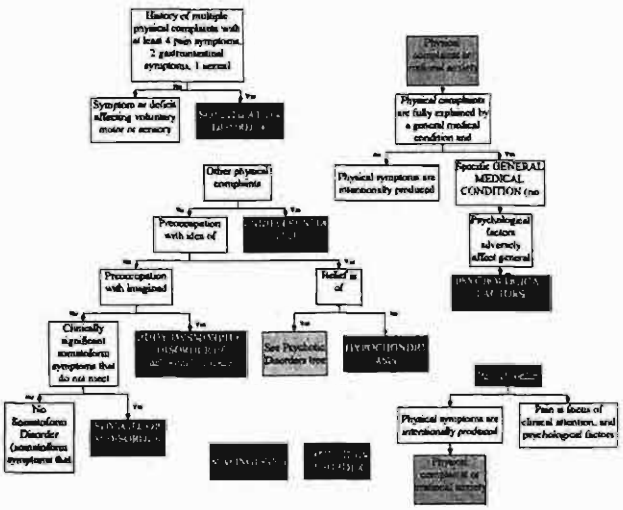


Fig. 9. The disconnected graph in Figure 1 laid out with the new algorithm (displayed with the same width to illustrate better usage of the area of aspect ratio 1.0).

Consequently, the grid step l can be calculated from the following quadratic equation:

$$(cn-1)\,l^2 - \sum_{i=1}^{n}(W_i + H_i)\,l - \sum_{i=1}^{n} W_i H_i = 0$$

With the average polyomino size $s \leq c$, the total area of all polyominoes does not exceed $n \cdot s$. Practical experiments show that the algorithm produces drawings of almost constant fullness (Figure 12), so the total packing area is also $O(n \cdot s)$.

To find a suitable place, each polyomino is tested for each cell. The test whether a polyomino fits in the specified place can be performed in $O(s)$ time in the worst case. Thus the complexity of the algorithm is $O(n^2 \cdot s^2)$. Since c and consequently s are constants this yields an $O(n^2)$ time overall, based on experimental results.

The value of the constant c must be selected carefully since its influence on the running time can be as much as $O(c^2)$. Figure 10 shows how the approximation quality paremeter c influences the adjusted fullness and the running time. For measurements, as a typical example a random forest of 300 trees of random order between 2 and 100 were generated (Figure 8 shows a smaller example of such a forest). The trees were laid out with a spring embedder algorithm similar to [10]. For small values of c, the adjusted fullness increases rapidly and converges towards 60%. The observed running time increases linearly with c. The difference from the theoretical bound of $O(c^2)$ can be explained by the observation that an occupied place is detected on average in constant time since almost all tested

places are occupied. The choice $c = 100$ seems to be a good compromise between the quality and the speed.

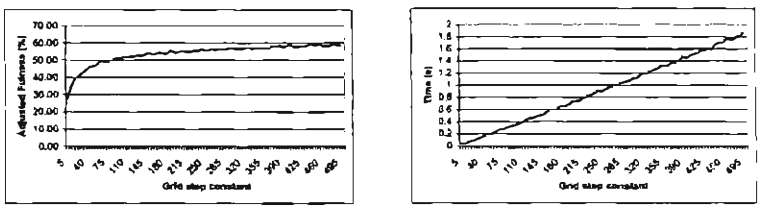


Fig. 10. How the grid step influences the adjusted fullness and running time.

The white space desired among the graph objects can be obtained by simply enlarging each object by half the spacing amount on each side.

The algorithm above does not favor one dimension over the other one and yields square-like drawings. In order to satisfy the desired aspect ratio DAR, one can simply take DAR as the unit grid step in x direction and 1 as the unit step in the y direction.

4.2 Packing in Multiple Pages

In certain applications, the graph is to be laid on multiple pages, and the task is to minimize the number of pages where the size of a page is defined in advance.

The approach is the same as above except the cost function is modified as $\max(x, y), x \geq 0, y \geq 0$, which defines the ordering starting from the corner of the page. Although such ordering lacks the nice central symmetry that we had in the original case, we have to modify the placement rule in order to better fill the sides of each page. We assume that each object separately fits in an empty page; otherwise an appropriate scaling should be performed. In algorithm PACKPOLYOMINOESINMULTIPAGES we start by fitting the first polyomino on the first page. If the current polyomino does not fit in the current page, the next page is tried until it is successfully placed. Similar to the original algorithm, the best results are obtained when the objects are sorted in their decreasing sizes. An optimization can be achieved by considering only those grid cells on the current page for which the bounding rectangle of the current polyomino is completely inside the page.

Here is a pseudo code of the multi page placement algorithm:

algorithm PACKPOLYOMINOESINMULTIPAGES($P_i, 1 \leq i \leq n$, *pageSize*)
(10) sort $P_i, 1 \leq i \leq n$ in the order of nonincreasing size
(11) initialize the grid G for the first page using *pageSize*
(12) **foreach** polyomino P_i **do**
(13) set *pageNo* to 1
(14) **while** P_i not placed **do**

```
(15)        calculate (x, y) such that the cost function is minimized
(16)        while cannot place P_i on page pageNo centered at (x, y)
(17)              and (x, y) is within page boundaries do
(18)           calculate next (x, y) using the cost function
(19)        end while
(20)        if P_i not placed then
(21)           set pageNo to the next one
(22)           if G not extended for page pageNo then
(23)              extend G for page pageNo
(24)           end if
(25)        end if
(26)     end while
(27)     mark those cells in G covered by P_i as occupied
(28)  end foreach
```

5 Comparison with Previous Methods

We have compared our new method with the tiling and alternate-bisection methods discussed earlier. During the experiments, graphs that contained up to a thousand disconnected objects were used. Each object was assumed to be a star polygon with random number of corners in $[3 \ldots 8]$, each with random integer coordinates in $[1 \ldots 100]$, all independent and uniformly distributed. The value for the approximation quality constant c was taken to be 100. The desired aspect ratio was taken to be 1. For the previous methods the tightest rectangles bounding these polygons were used, whereas with our new approach, the smallest polyomino tightly bounding the polygons were used. Figure 11 shows a sample set of drawings produced by these methods for the same set of objects. The performance comparison of the methods is presented in Figure 12.

Clearly the new approach results in much more compact drawings. In terms of the execution time, it is slower but still easily within acceptable bounds given the fact that it is highly rare that a graph contains more than a few hundred disconnected objects.

6 Conclusion

In this paper, we reviewed existing algorithms and presented a new approach for layout of disconnected graphs. The new approach uses polyominoes as opposed to rectangles used in previous approaches for representation of isolated nodes and components, and produces much more compact and uniform drawings. The parameters of our algorithm and how they affect the drawings produced as well as a variation of the algorithm for multiple pages were discussed.

Acknowledgement The authors wish to thank Cihad Baskoy for his help with the implementation and experimentation of certain algorithms.

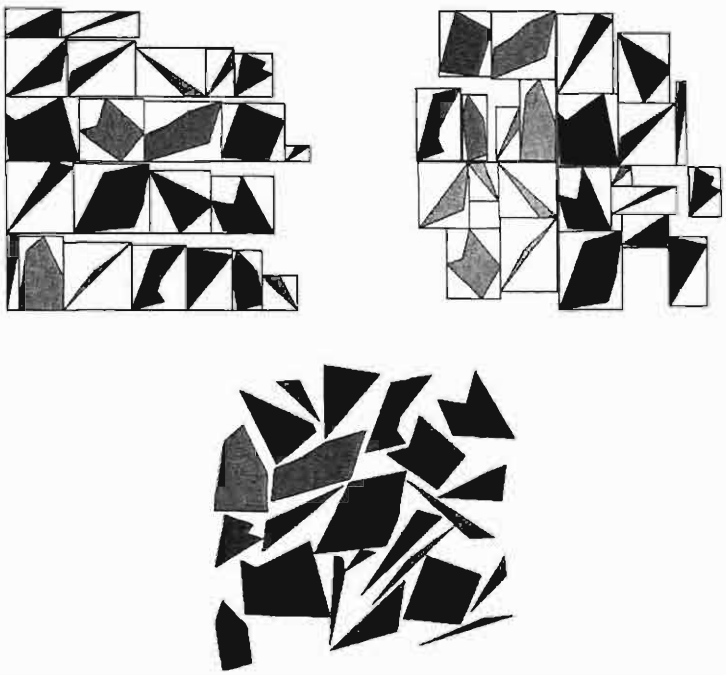


Fig. 11. A sample from the random set of objects laid out with all three methods: tiling (left), alternate-bisection (right), and polyomino (middle) packing.

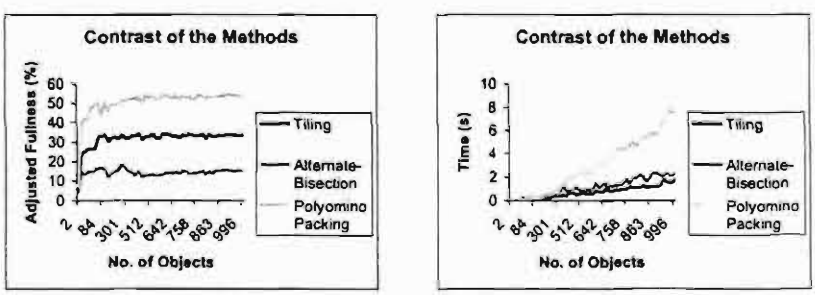


Fig. 12. Comparison of the new approach with the previous ones.

References

1. B. S. Baker, E. G. Coffman, and R. S. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, November 1980.
2. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49–106. Springer-Verlag, New York, 1984.
3. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, November 1990.
4. E. G. Coffman and P. W. Shor. Packings in two dimensions: Asymptotic average-case analysis of algorithms. *Algorithmica*, 9:253–277, 1993.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.
6. U. Dogrusoz. Algorithms for layout of disconnected graphs. *Information Sciences*, to appear.
7. U. Dogrusoz, M. Doorley, Q. Feng, A. Frick, B. Madden, and G. Sander. Toolkits for development of software diagramming applications. *IEEE Computer Graphics and Applications*, to appear.
8. U. Dogrusoz and G. Sander. Graph visualization. *ACM Computing Surveys*, to appear.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.
10. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
11. P. Kikusts and P. Rucevskis. Layout algorithms of graph-like diagrams of GRADE windows graphic editors. In F.J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 361–364. Springer-Verlag, 1995.
12. T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. John Wiley & Sons, 1990.

BIOINFORMATICS

Vol. 18 Suppl. 2 2002
Pages S202–S210

Building and analysing genome-wide gene disruption networks

J. Rung[1,†], T. Schlitt[1,†], A. Brazma[1], K. Freivalds[2] and J. Vilo[1]

[1]European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SD, UK and [2]Institute of Mathematics and Computer Science, University of Latvia, Riga, LV - 1459, Latvia

Received on April 8, 2002; accepted on June 15, 2002

ABSTRACT

Motivation: Microarray experiments comparing expression levels of all genes in yeast for hundreds of mutants allow us to examine properties of gene regulatory networks on a genomic scale. We can investigate questions such as network modularity, connectivity, and look for genes with particular roles in the network structure.

Results: We have built genome-wide disruption networks for yeast, using a representation of gene expression data as directed labelled graphs. Nodes represent genes and arcs connect nodes if the disruption of the source gene significantly alters the expression of the target gene. We are interested in features of the resulting disruption networks that are robust over a range of significance cutoffs. The networks show a significant overlap with analogous networks constructed from scientific literature. In disruption networks the number of arcs adjacent to different nodes are distributed roughly according to a power-law, like in many complex systems where the robustness against perturbations is important. The networks are dominated by a single large component and do not have an obvious modular structure. Genes with the highest outdegrees often encode proteins with regulatory functions, whereas genes with the highest indegrees are predominantly involved in metabolism. The local structure of the networks is meaningful, genes involved in the same cellular processes are close together in the network.

Keywords: gene networks, microarrays, yeast, graph visualisation

Availability: http://www.ebi.ac.uk/microarray/networks

Contact: schlitt@ebi.ac.uk; johan@ebi.ac.uk

INTRODUCTION

By measuring mRNA expression levels of thousands of genes in parallel, microarray experiments help revealing the structure of the underlying gene regulatory networks. The cellular regulatory system is a complex mechanism, and there is no straightforward way to represent it as a simple graphical model. The reduction of information needed to create a graphical model with nodes and arcs can be done in many ways, resulting in a wide range of network models with different interpretations.

A *gene network* is a directed labelled graph, where each node represents a gene and each arc represents a relation between the genes. The direction and the labels attached to an arc represent the nature and strength of the relation or the evidence for it. For example, an arc can mean that the source gene is coding for a transcription factor known to be binding to the promoter of the target gene. We obtain a different network if we define an arc as an observation in gene expression data, that the change in the expression level in the the source gene implies the change in the expression in the target gene. A network can also be built from literature data, linking genes which have been mentioned in the same paper (Stapley and Benoit, 2000; Jenssen *et al.*, 2001).

Various methods have been used to build gene expression networks, for instance, Bayesian networks (BN) (Friedman *et al.*, 2000) and Dynamic Bayesian networks (DBN) (Murphy and Mian, 1999). Pe'er *et al.* describe a method to infer subnetworks of interacting genes from gene expression data in a BN framework (Pe'er *et al.*, 2001). They are interested in finding features with high confidence in BNs learnt from a set of 565 genes in mutation experiments for *Saccharomyces cerevisiae* (Hughes *et al.*, 2000). Because the data set is too small to yield a single high confidence network, they use bootstrapping techniques to find such features. These are extracted into subnetworks, which are interpreted as separate cellular processes or putative interactions.

Although BN based methods are powerful techniques, the approaches that have been implemented currently are only able to deal with relatively small subsets of genome data. We have chosen an approach that allows for a genome-wide analysis, and although simple, we can demonstrate that it is biologically meaningful and provide insight in the genome-wide organisation of the gene networks. We build gene disruption networks by

†These authors contributed equally to this paper.

ollecting information about differences in gene ex-ression between yeast mutant strains. Nodes represent enes and arcs connect the deleted genes with the genes or which expression level changes have been observed or a particular threshold. We compare these networks o a network derived from literature data and explore heir properties, including the distribution of arcs and he cellular functions of genes with many interactions as well as the topology and robustness of the network for different discretisation thresholds. Finally, we present an example of a subnetwork including genes involved in the pheromone response pathway.

BUILDING THE GENE DISRUPTION NETWORKS

A *directed graph* is defined as a tuplet $(\mathcal{G}, \mathcal{A})$ of nodes $\mathcal{G}$ and arcs $\mathcal{A}$, where an arc $a \in \mathcal{A}$ is an ordered pair of nodes $(g_1, g_2) \subseteq \mathcal{G}$. We can attach labels to nodes and arcs, in which case we obtain a *labeled graph*. In a *gene disruption network* Δ, each node represents a gene. A gene g_1 is connected to gene g_2 with an arc $a = (g_1, g_2)$, if the disruption of the gene g_1 changes the expression level of gene g_2 significantly. The 'significance' is defined through a threshold as described below.

The starting point for our network building is a *gene expression data matrix* E, in which each experimental condition, which in our case is a disruption of a particular gene, corresponds to a column and each gene corresponds to a row. The jth element in the ith row holds the 'expression level' r_{ij} of gene i in experiment j, more precisely $r_{ij} = \log(l_{ij}/c_{ij})$, where l_{ij} is the background corrected signal for the studied condition relative to that of the same gene in a control c_{ij}.

Next, we transform the original gene expression data matrix E into a discretised matrix D, where values $d_{ij} \in \{-1, 0, 1\}$ represent the expression level being decreased, unchanged or increased with respect to the control experiment. For this we first normalise the log-ratios r_{ij} to $\tilde{r}_{ij} = r_{ij}/\hat{\sigma}_{ij}$, where $\hat{\sigma}_{ij}$ is a gene-specific standard deviation estimate for the measurement of gene i under experimental condition j. This can be estimated using different error models, see for instance (Hughes *et al.*, 2000). The discretisation is carried out on the normalised data using a cutoff level $\gamma > 0$, and defining

$$d_{ij} = \begin{cases} -1, & \tilde{r}_{ij} \leq -\gamma \\ 0, & -\gamma < \tilde{r}_{ij} < \gamma \\ 1, & \tilde{r}_{ij} \geq \gamma. \end{cases} \quad (1)$$

Due to the gene-specific normalisation, we can use one consistent cutoff level for all measurements.

Effectively the disruption network $\Delta(\gamma) = (\mathcal{G}, \mathcal{A})$ is a representation of the discretised matrix as a graph. For the given cutoff γ, we draw an arc from gene $g_j \in \mathcal{G}$ to gene $g_i \in \mathcal{G}$, if $d_{ij} \neq 0$. We can label the arcs as downregulating or upregulating, depending on whether $d_{ij} = -1$ or $d_{ij} = +1$, respectively. In the graphical representation we can show this by drawing them in red and green (or solid and gray in black-and-white representation). We also label the nodes by their respective gene names. Examples of gene disruption networks are given in Figure 4 and Figure 5.

We use the microarray data set from Hughes *et al.* (Hughes *et al.*, 2000), which includes expression profiles for all genes in *Saccharomyces cerevisiae* over a set of 300 experiments. In 274 experiments single gene deletion mutants were examined, 2 experiments were double gene deletion mutants, 13 experiments were done with genes with tetracyclin regulated promoters and in the remaining 11 experiments the yeast cell cultures were treated with different drug compounds.

In parallel to these experiments, a series of 63 control experiments were performed, comparing untreated wild-type yeast cultures to each other, permitting the use of a gene specific error model and to normalise the data using the standard deviation estimates. Following normalisation we discretised the data matrix using different significance cut-offs between $\gamma = 1.0$ and $\gamma = 26.0$, which was the threshold where no edges at all were found. However, the analyses described below concentrate on the range $\gamma = 2.0, 2.1, \ldots, 4.0$. We included only genes with less than 25% undefined data points in the gene expression matrix as given in (Hughes *et al.*, 2000).

NETWORK VISUALISATION

The analysis of the network properties is facilitated by their visualisation based on a graphical layout in 2 dimensions. The gene disruption networks may be very large and dense, therefore their visualisation is not trivial. To visualize the obtained networks we used the Graphical Diagramming Engine (GDE) (Freivalds and Kikusts, 2001). GDE supports five layout styles, each revealing a different aspect of the network. In the hierarchical style drawings it is easy to notice the nodes with high in- and out-degree and their relationships. The spring-embedder layout reveals the cluster structure of the network, clearly identifying the parts that are strongly related. The grid layout was used to produce the drawings in this paper; it gives the most compact drawings useful when the area for displaying the drawing is limited. The tool also provides several edge representations including orthogonal, polyline and spline. Here the spline representation was the most adequate since the smoothness allows easier following of the edge. Although layouts of these networks were generated automatically, the rich editing facilities powered by the quadratic optimization method (Freivalds and Kikusts, 2001) were used in exploring the networks, discovering their features and preparing their relevant parts for pre-

sentation. For connected component placement an original polyomino packing algorithm (Freivalds *et al.*, 2001) was used. Such an approach allows a compact placement and high visual separation of the components.

COMPARISON OF DISRUPTION NETWORKS WITH A LITERATURE NETWORK

In order to relate the disruption networks to biological results, we compared them to a reference network constructed using the curated database YPD (Costanzo *et al.*, 2001), which contains information about the yeast genome.

A *YPD network* is a graph, where the nodes represent genes and two genes are connected with an undirected edge,if gene X is mentioned in the description of gene Y in the YPD database. Thus an edge between X and Y could be read as 'the database entry for X mentions Y and/or vice versa'. Note that we have not done any further analysis of the syntax, e.g. if the original description was 'a change in x does not influence y' we would still have an edge. These cases are rare and do not diminish the overall use of the reference network. The lack of text analysis is also the reason why we use undirected edges instead of directed edges, e.g. if the description for gene X reads 'X regulates Y' then there is also a relation 'Y is regulated by X'.

For our study, we selected a subset of 274 genes studied in (Hughes *et al.*, 2000). The resulting network has 827 edges. If our disruption networks represent meaningful biological knowledge, we would expect that it should overlap with the YPD network more than expected by chance. We checked this assumption by calculating the overlap between the disruption networks and the reference network, then comparing it to the distribution of overlaps between randomised networks and the reference network. Each randomised instance of the disruption network was built by keeping the number of arcs present in the disruption network constant. Every node has on average the same in- and outdegree as in the original network. This procedure assures that the overall network topology is retained.

We created 1000 randomised networks for each disruption network built. All disruption networks showed a significantly higher overlap with the reference network than the corresponding randomised networks (see Figure 1). In the range of $\gamma \in \{2.0, 2.1, \ldots, 4.0\}$, the overlap between the disruption networks and the reference network increases monotonously from 9–16%. If we interpret the reference network as representing connections between genes reported in literature, this result demonstrates that the disruption networks contain significantly more biological information than achieved randomly.

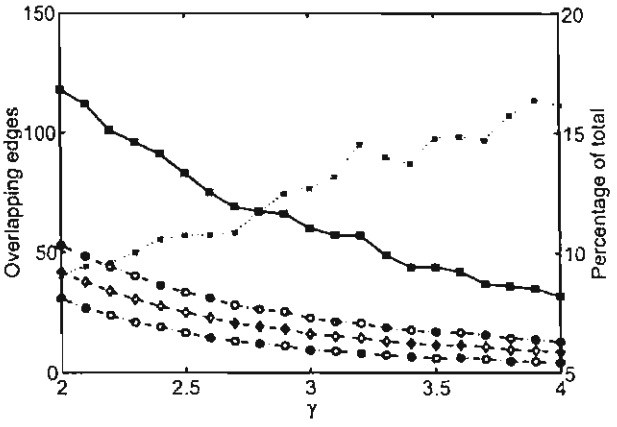


Fig. 1. Disruption networks compared with randomised ones using a reference network based on YPD data. The solid line shows the number of arcs which are found in both the disruption network for cutoff level γ and the YPD network. The dashed line shows the average overlap between 1000 randomised networks and the same YPD network, with 2 standard deviations shown by the dash-dotted line. The dotted line shows the percentage of edges in the disruption networks that are also present in the YPD network. For the range of γ studied, this statistic is monotonously increasing and no obvious optimisation can be done based on it.

'IMPORTANT' GENES AND GENES WITH COMPLEX REGULATION

The *degree of a node in a graph* is defined as the number of adjacent edges. In directed graphs we distinguish between the *indegree*—the number of incoming arcs—and the *outdegree*—the number of outgoing arcs. For disruption networks we can speculate that genes with a high outdegree are 'important' in the sense that they influence the expression of many other genes, while genes with a high indegree have a complex regulation mechanism. In order to analyze the indegree and outdegree of various genes we represent them in two different formats: the *degree table*, where the genes are sorted according to their outdegree or indegree, together with their annotation in the YPD database (Costanzo *et al.*, 2001); the *cellular role table*, where we use the 'cellular role' annotation from YPD to group all genes with at least one arc in a particular network and calculate for each group the median indegree and outdegree, and sort the groups according to their median degree.

We find that the distribution of total degree, the sum of the in- and out-degree for each node, roughly follows a power-law (Figure 2). This topology, denoted *scale-free* (Barabasi and Albert, 1999), has been found for metabolic networks (Jeong *et al.*, 2000) and for many other complex systems (Albert and Barabasi, 2002).

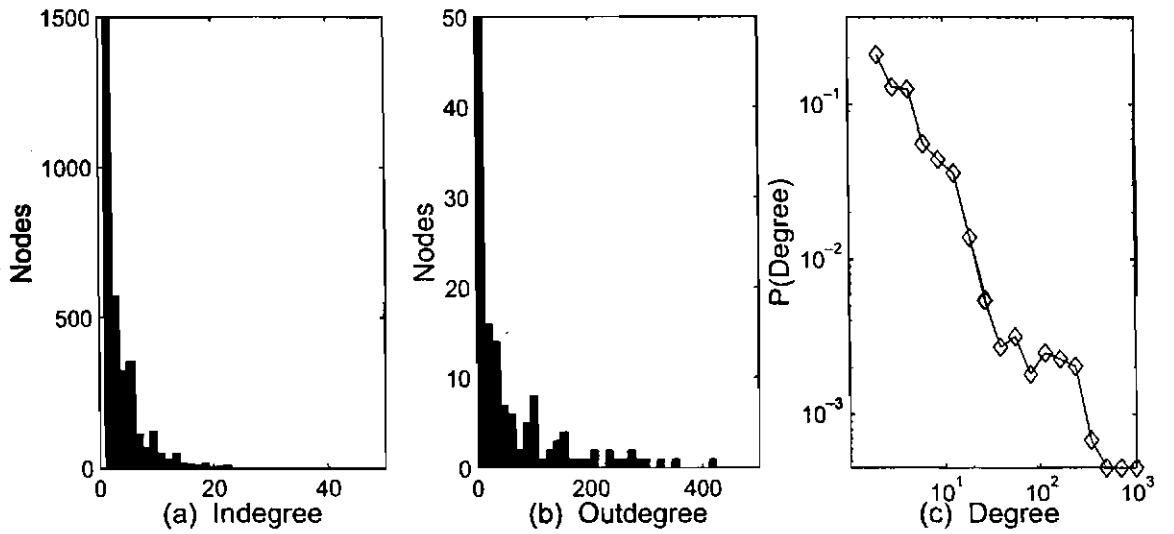


Fig. 2. (a,b) Histogram of number of nodes (a) with particular indegree or outdegree (b) for $\gamma = 2.5$. (c) Log–log plot of the distribution of total degree for the disruption network with $\gamma = 2.5$

The genes with the highest outdegrees encode proteins involved in transcriptional regulation (TUP1, SHE4, SWI4, CYC8)†, a ribosomal protein (RPL12A), a protein involved in rRNA modification and degradation of aberant mRNA (RRP6), a histone deactylase (RPD3), a MAP kinase (KSS1), proteins involved in metabolism (ANP1, ERG2, FKS1) and a H^+-ATPsynthase subunit (CUP5). It is quite remarkable that in the case of the TUP1 deletion mutant about 50% of the genes show changes in expression and still the yeast cells survive. A full degree table for $\gamma = 2.0$ is included in our supplementary data.

The order of the groups in the cellular role table is robust over a wide range of cut-off values (see Table 1), though the number of genes per group is relatively small since only 248 mutants were included in the data set. The cellular roles with the highest median outdegrees predominantly have regulatory functions. It has to be kept in mind that genes can belong to more than one of these groups and sometimes the annotation can be misleading. RTG1 for example has an outdegree of 316, it is a transcription factor, but it is also a member of the 'carbohydrate metabolism' group.

The overwhelming majority of genes with the highest indegree are involved in metabolism (ADE17, VID24), especially amino acid biochemistry (YGL009C, HIS5, HIS1, TWT1, HOM3), stress response (HSP12, YHB1), transport (SIT1, PEX21, ARN1, YHM1) as well as some genes of unknown function (see supplementary data). The ordering of the groups in the cellular role table for the indegree is robust over a wide range of cut-off values. The group sizes are much larger compared to the cellular role table for the outdegree (see Table 1), therefore the grouping is more reliable.

Only a few genes can be found to have a high indegree and a high outdegree at the same time. When we plot the genes using their rank for outdegree and indegree in the degree table we find that ARG5,6 with an outdegree of 108 and an indegree of 28 (see Figure 3) is the only gene which is within the top 50% of the genes with highest outdegree and highest indegree.

The comparison of the cellular roles of the genes with the highest out- and in-degrees in the yeast mutation networks seem to confirm the intuition that general regulators are influencing many genes, whereas some metabolic genes are being regulated by many other genes. Note that the only group having a high median indegree and outdegree is 'small molecule transport'.

CONNECTED COMPONENTS

Recently, some investigations on the overall structure of gene networks have been published. Wagner predicts the existence many independent subnetworks and only a few direct connections for each gene (Wagner, 2002), whereas Featherstone and Broadie argue that there is 'a single giant functional component rather than several subnetworks'. They also found that hubs, the genes with highest degrees, are evolutionary more conserved than the genes with lower degrees. (Featherstone, 2002).

† We use capital letters to refer to genes in the disruption networks.

Table 1. Cellular role table showing the top 5 groups with the highest median degrees for selected networks, with a minimum group size of 3 for outdegree and 40 for indegree (m: the median degree, n: the group size)

γ	outdegree	m	n	indegree	m	n
2.0	carbohydrate metabolism	363	4	amino-acid metabolism	9	194
	RNA turnover	**353**	**4**	nucleotide metabolism	**6**	**82**
	meiosis	**244**	**3**	energy generation	5	**242**
	cell stress	**207**	**9**	*small molecule transport*	5	343
	protein translocation	**197**	**3**	other metabolism	5	148
2.8	RNA turnover	**110**	**4**	amino-acid metabolism	4	167
	cell stress	**62**	**8**	nucleotide metabolism	3	67
	meiosis	**54**	**3**	energy generation	2	184
	protein synthesis	**53**	**7**	differentiation	2	43
	cell wall maintenance	**47**	**6**	*small molecule transport*	2	286
3.6	RNA turnover	**48**	**4**	*small molecule transport*	2	230
	RNA processing/ modification	**41**	**4**	other metabolism	2	96
	cell stress	**27**	**8**	nucleotide metabolism	2	58
	small molecule transport	**19**	**8**	mating response	2	57
	cell wall maintenance	**19**	**6**	amino-acid metabolism	2	133

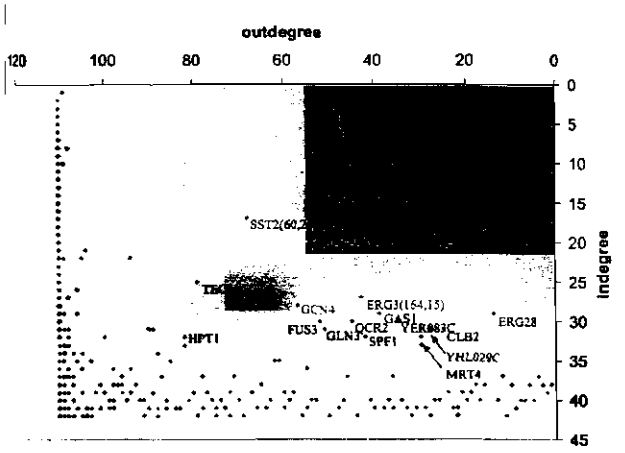


Fig. 3. Genes plotted according to their rank in the indegree and outdegree table for the network with $\gamma = 2.0$. Some outdegrees and the indegrees are given in brackets. The shaded areas indicate regions of ranking, darkest grey top 50%, medium grey top 66%, light grey top 75%. The only gene with a in- and out-degree within the top 50% for both is ARG5,6, a acetylglutamate kinase and N-acetyl-gamma-glutamyl-phosphate reductase.

To address the open question about modularity, we are interested in the connectivity structure of our networks and want to examine whether or not, and to what extent yeast disruption networks have a modular structure. A *connected component* in a graph is defined as a subgraph where there exists a path through arcs (ignoring the direction) leading from one node to the other for each pair of nodes. A single gene disconnected from all other genes is not regarded as a separate connected component. For a low enough threshold γ all genes will be connected.

For $\gamma \leq 3.0$ only one connected component can be found[‡]. For higher thresholds γ we find one dominant and few small components of sizes 2 or 3 genes. The dominant connected component consists of 5383 genes for $\gamma = 2.0$ and of 2354 genes for $\gamma = 4.0$ (see table 2).

Since the networks contain only a small number of nodes with rather high degrees, it is interesting to see if we can break down the connected component by removing the genes with the highest indegree or outdegree from the networks. For this we removed the top 1, 5 and 10% of genes when ranked for the outdegree, respectively indegree. This disrupts the connected component considerably. However, for $\gamma \leq 3.6$ we still find only one major component and some much smaller components even when removing the top 10% of the genes with highest degrees (see Table 2). The dominant components are reduced in size, but are still at least one order of magnitude bigger than the minor components.

Only the networks with $\gamma \geq 3.7$ consist of components of roughly equal sizes when the 10% of the genes with highest degrees are removed. Removing the top ten percent of the genes with the highest degrees at $\gamma = 3.7$ yields a network with 378 genes with at least one adjacent arc and a total of 331 arcs. The five biggest components have size 93, 53, 31, 20, 10 and there is a total of 50 components. Although in some of these components genes belonging to the same YPD annotation group are overrepresented, this is not true for the majority of the

[‡] There is an exception for the network at $\gamma = 2.6$, which has an additional component consisting of 2 nodes

Table 2. Size of the biggest two and the total number of connected components in selected networks, the column 'full' refers to the original network, the numbers in the other columns result from removing the genes with the highest degrees (1, 5, 10%: proportion of genes which were removed)

γ		full	removed 1%	removed 5%	removed 10%
10	biggest	5383	4707	3682	2614
	second			2	5
	total number	1	1	2	2
10	biggest	3556	2461	1385	764
	second	2	2	4	6
	total number	2	2	9	17
16	biggest	2789	1612	901	485
	second	2	3	10	11
	total number	3	4	13	26
17	biggest	2675	1497	825	93
	second	2	3	9	53
	total number	3	4	14	50
10	biggest	2354	1205	542	45
	second	3	3	6	28
	total number	4	7	22	51

groups. It seems possible that these components may not have a true biological meaning, but are rather the result of the mechanical reduction of the graph.

Modules are thought to be building blocks which confer a particular function and might be interlinked to other modules via hubs. However, the results of Featherstone *et al.* and our analysis would argue for a closely connected network organisation rather than a modular structure. Genes involved in the same cellular processes seem to be closer in the network than unrelated genes (see also next section), however, this does not mean that the network can be easily broken down into independent parts. This indicates a scale free structure, which has the property of self-similarity, meaning that any part of the network is statistically similar to the whole network (Wolf *et al.*, 2002).

SUBNETWORKS AND THEIR VISUALISATION

Finally, we focus on subnetworks containing genes of particular interest, as an example taking the pheromone response. When yeast cells are starved they undergo meiosis leading to the production of haploid spores. Spores give rise to haploid cells of two different kinds of mating types a and α. These haploid cells can divide by budding, but when mixed they are able to mate and form diploid cells. Many changes are involved in the switch from haploid cells to diploid cells. The haploid cells produce a mating type specific pheromone, which diffuses into the medium

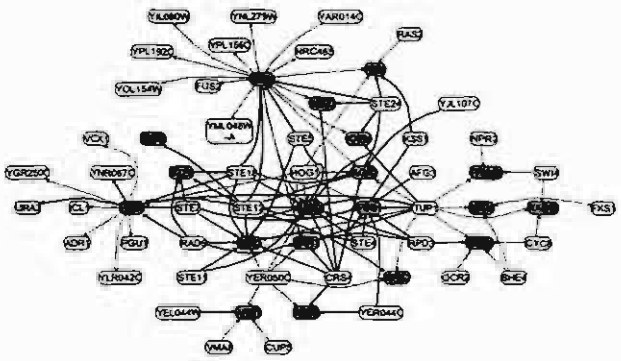


Fig. 4. This subnetwork is the result of filtering the full network at $\gamma = 4.0$ for the core set marked in grey and their next neighbours (grey arcs: downregulation, dark arcs: upregulation). See web supplement for a coloured figure (http://www.ebi.ac.uk/microarray/networks)

and attracts mating partners of the opposite mating type. Diploid cells neither produce pheromone nor are they responsive to it. A G-protein coupled receptor system is involved in pheromone sensing in haploid cells and is one of the best studied signal transduction pathways in biology. The pheromone response includes increased transcription of genes whose products facilitate mating, arrest of the mitotic cell division cycle, changes in the cell surface and nucleus for fusion with the cognate organelles of the mating partner and alterations of the cell polarity and morphology (Sprague and Thorner, 1992).

We filtered for a *core set* of 20 genes known to be involved in pheromone response and their next neighbours in the disruption network. The subnetwork contains 63 genes and 115 arcs for $\gamma = 4.0$ (see Figure 4), 36 genes are adjacent to more than one arc. Of these 36 genes 18 genes belong to the core set and further 8 genes (STE4, STE5, STE7, STE11, STE12, STE18, STE24, KSS1) are annotated as involved in the mating response. Some of the remaining 10 genes are likely to encode proteins which are involved in processes that are related to the pheromone response. One example is HOG1, which encodes a MAP kinase like STE11, however this protein is involved in the high-osmolarity signal transduction pathway, which is similar to the pheromone response pathway. Another example is SHE4, which encodes a protein required for the duplication of the spindle pole body. Finding these groups could possibly be done by appropriate clustering. However, our method allows also to find genes with high outdegrees (SWI4, CYC8, TUP1, RPD3) in the subnetwork, which encode gene products known to be members of general regulatory complexes which influence many genes. The remaining four genes are RAD6, CRS4, YER044C and YER050C.

The same filtering at $\gamma = 2.0$ gives us a subnetwork with 240 genes and 470 arcs (see Figure 4), 75 genes have a degree higher than 1, including the full core set and the remaining genes mentioned in the previous paragraph. Additionally we find more genes encoding proteins which are likely to be involved in pheromone response related processes. For instance, BNI1 is required for the bipolar budding pattern, RTT104 is a helicase important for replication of ribosomal DNA, RAS2 is a GTP binding protein involved in regulation of the cAMP pathway, KIN3 is a serine/threonine kinase, FKS1 and GAS1 are involved in the cell wall synthesis.

The transcription factor Ste12 is the final protein in the pheromone signal transduction cascade. A consensus sequence for the DNA binding site of Ste12 is known ([A]TGAAACAA), and many genes involved in pheromone response have more than one Ste12-binding site in their promoter (Sprague and Thorner, 1992). Of 19 genes known to be induced by pheromone 11 are found to be connected to STE12 in the network for $\gamma = 2.0$ (10 for $\gamma = 3.0$), while only 2 out of 13 genes known not to respond to the pheromone signal, are connected to STE12 in the network with $\gamma = 2.0$ (none for $\gamma = 3.0$). We used PATMATCH[§] to locate the consensus sequence for the binding sites in the upstream regions of the genes (Vilo and Kivinen, 2001). The connectivity to STE12 in our networks corresponds to the absence or presence of the consensus binding site in the promoter regions of the corresponding genes (see supplementary data).

We tested several core sets containing genes involved in mating response with similar results. Up to 77% (depending on γ threshold) of the genes adjacent to the core set are annotated as involved in mating response as well.

We also tested core sets containing genes involved in other processes than mating response. These sets, e.g. small molecule transport, signal transduction and cell wall synthesis showed a similar behaviour, however fewer genes adjacent to the core set were involved in the same cellular process (up to 27% for small molecule transport, 20% for signal transduction, 21% for cell wall synthesis). For some core sets we found very few of the adjacent genes belonging to the same annotation group, e.g. energy generation with only up to 5% genes of the adjacent genes in the same group.

DISCUSSION

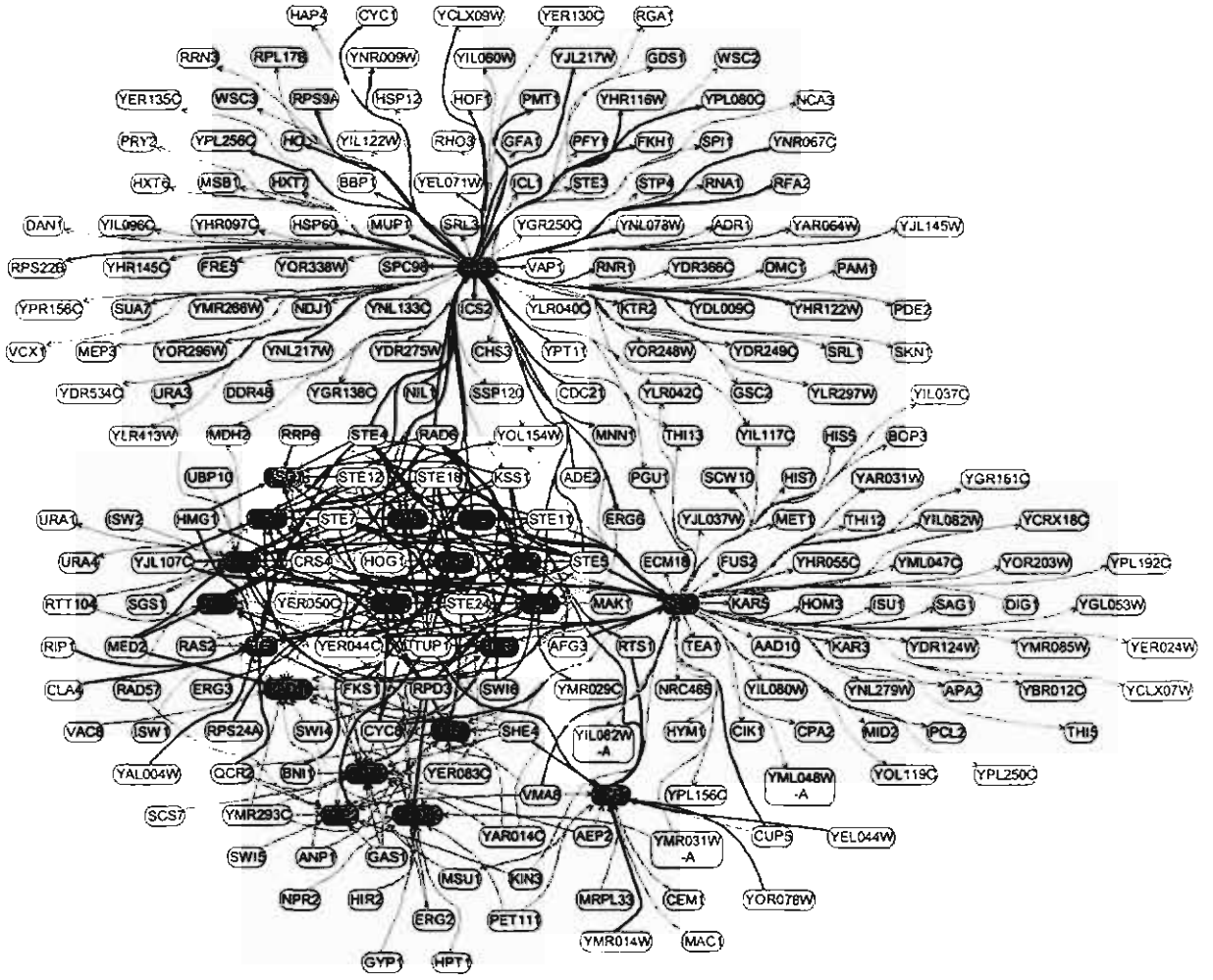
We studied gene disruption networks for yeast, inferred from gene expression data fully automatically without any human intervention. Yet, the overlap between these networks and the reference network constructed from the YPD database is always considerably higher than expected by chance. This encouraged us to study the properties of the disruption networks further.

We have studied structural features of these networks that are robust for a range of cutoff thresholds. We notice that the distributions of the numbers of incoming and outgoing arcs are rather uneven, with few genes having a large number of incoming or outgoing arcs, while most genes had very few of each. The genes with the highest number of outgoing arcs can be regarded as being 'important' for cellular regulation, and it is encouraging that the annotation of such genes indicate their regulatory functions. The genes with the highest number of incoming arcs can be regarded as having complex regulation, and again, it is encouraging that these genes typically have a metabolic function according to their annotations. 'Small molecule transport' was the only class of genes that appeared in both classes.

We found that regardless of the cutoff threshold the disruption networks have only one clearly dominant connected component with rather small components of one to three genes being separated for higher significance threshold. Generally the same property stands if we remove the genes with the largest number of indegree or outdegree (these genes could be potentially be the ones holding the network together). The only case where the network falls into a number of subnetworks of relatively equal size is for very strong perturbations, and the disconnected componenents found when enforcing the network to break up this way do not have obvious biological meaning.

The network topology is dominated by this large connected component, and has a distribution of arcs which roughly follows a power-law. It has been suggested that this network topology is generated by a system which is optimised to work under conditions where it has to be robust against perturbations, but where this tolerance has a cost (Carlson and Doyle, 1999). This is typical for biological networks, which have gone through natural selection to be tolerant to uncertain environments. A biological system cannot be protected against all possible threats, since the cost would be too high. Therefore a topology is favoured where a disruption of one component is most likely to affect few others, but where it is unlikely to disrupt a central component which may cause severe damage or death. For instance protein networks (Jeong *et al.*, 2001) exhibit this type of robustness. Our findings support this theory since the data used is specifically information about gene disruptions. Intimately linked with this is the still open question about modularity in gene networks. For instance, Featherstone *et al.* argue that a network with a scale-free structure will be dominated by a single large component. Our results support this theory, since we were not able to find any discrete modules in our networks. However, it may be

[§]For PATMATCH and other tools within Expression Profiler see http://ep.ebi.ac.uk.



ig. 5. This subnetwork is the result of filtering the full network at $\gamma = 2.0$ for the core set marked in grey and their next neighbours (grey ·es: downregulation, dark arcs: upregulation). See web supplement for a coloured figure (http://www.ebi.ac.uk/microarray/networks)

ossible to find such modules by examining the networks ı more detail, for instance by looking for the smallest cuts ı the network graph which lead to disconnectivity and exmine whether the resulting components have a biological ıeaning.

Looking for subnetworks consisting of genes involved ı a particular cellular process allowed us to predict genes ·ith similar functional roles. We need further studies to ee how widely such predictions can be generalised.

Finally note that we can view the building of gene ısruption networks as a method of structuring gene ·pression data which is an alternative to other known ıethods such as hierarchical clustering. Such networks llow us to explore different aspects of gene expression ata.

ACKNOWLEDGEMENTS

We thank Frank Holstege, Patrick Kemmeren, Sabine Dietmann, Laurence Ettwiller and Michael Lappe for valuable discussions and Helen Parkinson for proofreading of the manuscript. JR acknowledges support from the AIM graduate school, Uppsala University, the Swedish Science Research Council (VR) and TTA Technotransfer AB.

REFERENCES

Albert,R. and Barabasi,A. (2002) Statistical mechanics of complex networks. *Rev. Mod. Phys.*, **74**, 47.

Barabasi,A. and Albert,R. (1999) Emergence of scaling in random networks. *Science*, **286**, 509.

Carlson,J. and Doyle,J. (1999) Highly optimzed tolerance: a mechanism for power laws in designed systems. *Phys. Rev. E*, **60**, 1412–1427.

Costanzo,M., Crawford,M., Hirschman,J., Kranz,J., Olsen,P., Robertson,L., Skrzypek,M., Braun,B., Hopkins,K., Kondu,P., Lengieza,C., Lew-Smith,J., Tillberg,M. and Garrels,J. (2001) YPD, PombePD and WormPD: model organism volumes of the BioKnowledge library, an integrated resource for protein information. *Nucleic Acids Res.*, **29**, 75–79.

Featherstone,D.E.B.K. (2002) Wrestling with pleiotropy: genomic and topological analysis of the yeast gene expression network. *Bioessays*, **24**, 267–274.

Freivalds,K., Dogrusoz,U. and Kikusts,P. (2001) Disconnected graph layout and the polyomino packing approach. *Proc. of Graph Drawing*, in print

Freivalds,K. and Kikusts,P. (2001) Optimum layout adjustment supporting ordering constraints in graph-like diagram drawing. *Proc. Latvian Acad. Sci.*, **55**, 43–51.

Friedman,N., Linial,M., Nachman,I. and Pe'er,D. (2000) Using bayesian networks to analyze expression data. *RECOMB 2000*.

Hughes,T.R., Marton,M.J., Jones,A.R., Roberts,C.J., Stoughton,R., Armour,C.D., Bennett,H.A., Coffey,E., Dai,H., He,Y.D., Kidd,M.J., King,A.M., Meyer,M.R., Slade,D., Lum,P.Y., Stepaniants,S.B., Shoemaker,D.D., Gachotte,D., Chakraburtty,K., Simon,J., Bard,M. and H.,F.S. (2000) Functional discovery via a compendium of expression profiles. *Cell*, **102**, 109–126.

Jenssen,T.L., greid,A., Komorowski,J. and Hovig,E. (2001) A literature network of human genes for high-throughput analysis of gene expression. *Nature Genet.*, **28**, 21–28.

Jeong,H., Mason,S.P., Barabasi,A.L. and Oltavai,Z.N. (2001) Lethality and centrality inprotein networks. *Nature*, **411**, 41.

Jeong,H., Tombor,B., Albert,R., Oltavai,Z.N. and Barabasi,A.L. (2000) The large-scale organization of metabolic networks. *Nature*, **407**, 651–654.

Murphy,K. and Mian,S. (1999) *Modelling Gene Expression Data Using Dynamic Bayesian Networks*, Technical Report, U.C. Berkeley, Department of Computer Science.

Pe'er,D., Regev,A., Elidan,G. and Friedman,N. (2001) Inferring subnetworks from perturbed expression profiles. *Bioinformatics* suppl 1, *ISMB 2001*, **17**, 215–224.

Sprague,G.J. and Thorner,J. (1992) Pheromone response and signal transduction during mating process of *Saccharomyces cerevisiae*. *The Molecular and Cellular Biology of the yeast Saccharomyces*, Volume 2 of *Monograph Series-21*, Jones,E., Pringle,J. and Broach,J. (eds), Cold Spring Harbour Laboratory Press, pp. 657–744.

Stapley,B. and Benoit,G. (2000) Biobibliometrics: information retrieval and visualisation from co-occurrences of gene names in medline abstracts. *Pac. Symp. Biocomput.*, **5**, 526–537.

Vilo,J. and Kivinen,K. (2001) Regulatory sequence analysis: application to the interpretation of gene expression. *Eur. Neuropsychopharmacol.*, **11**, 399–411.

Wagner,A. (2002) Estimating coarse gene network structure from large-scale gene perturbation data. *Genome Res.*, **12**, 309–315.

Wolf,Y., Karev,G. and Koonin,E. (2002) Scale-free networks in biology: new insights into the fundamentals of evolution? *Bioessays*, **24**, 105–109.