

On labeling in graph visualization [☆]

Ugur Dogrusoz ^{b,a,*}, Konstantinos G. Kakoulis ^c, Brendan Madden ^a,
Ioannis G. Tollis ^{d,e}

^a *Tom Sawyer Software, Oakland, CA, USA*

^b *Computer Engineering Department, Bilkent University, Cankaya, Ankara 06800, Turkey*

^c *National Organisation for Medicines, Cholargos, Greece*

^d *Computer Science Department, University of Crete, Greece*

^e *Institute of Computer Science, Hellas-FORTH, Crete, Greece*

Received 17 January 2007; accepted 17 January 2007

Abstract

When visualizing graphs, it is essential to communicate the meaning of each graph object via text or graphical labels. Automatic placement of labels in a graph is an NP-Hard problem, for which efficient heuristic solutions have been recently developed. In this paper, we describe a general framework for modeling, drawing, editing, and automatic placement of labels respecting user constraints. In addition, we present the interface and the basic engine of the Graph Editor Toolkit – a family of portable graph visualization libraries designed for integration into graphical user interface application programs. This toolkit produces a high quality automated placement of labels in a graph using our framework. A brief survey of automatic label placement algorithms is also presented. Finally we describe extensions to certain existing automatic label placement algorithms, allowing their integration into this visualization tool.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Graph visualization; Graph labeling; Automatic label placement

1. Introduction

Graphs model many types of relational information encountered in numerous areas. Examples vary from UML diagrams to PERT charts, from network maps to business process design diagrams, to biological pathways. Graph drawing, or layout, is the positioning of nodes (objects) and the routing of edges (relations) in a graph in order to produce an aesthetically pleasing, comprehensible drawing of the underlying data [4,20]. Visualizing such graphs has become essential in software engineering, telecommunications, enterprise software and electronic commerce, knowledge management, bioinformatics, and financial analysis.

[☆] Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162. A US Patent (Number 6,091,424) has been received by Tom Sawyer Software on some of these and related results.

* Corresponding author. Present address: Computer Engineering Department, Bilkent University, Cankaya, Ankara 06800, Turkey. Tel.: +90 312 290 1612; fax: +90 312 266 4047.

E-mail address: ugur@cs.bilkent.edu.tr (U. Dogrusoz).

In many diagramming applications, it is essential that a drawing be labeled. Labels assist in conveying information or clarifying the meaning of complex structures, and can be classified into three: edge labels, node labels, and graph labels. Figs. 1–4 show examples of diagrams with labels. The problem of positioning labels corresponding to graphical objects of drawings is called *automatic label placement* or simply *graph labeling*.

Just as graph layout is a time-consuming and monotonous task, so is the automatic placement of labels. The graph labeling problem falls into the class of NP-Hard problems [16,22]. Recent advances offer efficient solutions to the problem [3,11,17]; for a comprehensive bibliography on labeling, see [25].

Many graph layout and editing systems have been developed in the past. Please refer to [5,7] for an overview of such systems. One essential aspect has not been addressed much; this is the ability to support the automatic placement of labels in a drawing. Only recently some tools have integrated such capabilities [13,14,26].

The Graph Editor Toolkit (GET) [13,7] is a family of graph editing and layout libraries that facilitate easy integration with graphical user interface programs for the development of applications that require diagramming visual interfaces. Graph layout comes in different styles, each having particular features and benefits suited to different industries and applications. The Graph Editor Toolkit offers four different layout libraries: *Circular*, *Hierarchical*, *Orthogonal*, and *Symmetric*. Each library in the Graph Editor Toolkit is equipped with fast algorithms for automatic placement of labels which are based on [16,15,18].

In this paper, we present a framework for modeling, drawing, and automatic placement of labels as implemented in the Graph Editor Toolkit. Furthermore, some of the challenges of incorporating a labeling interface into a graph layout system, and how to resolve these within the architecture of GET are discussed.

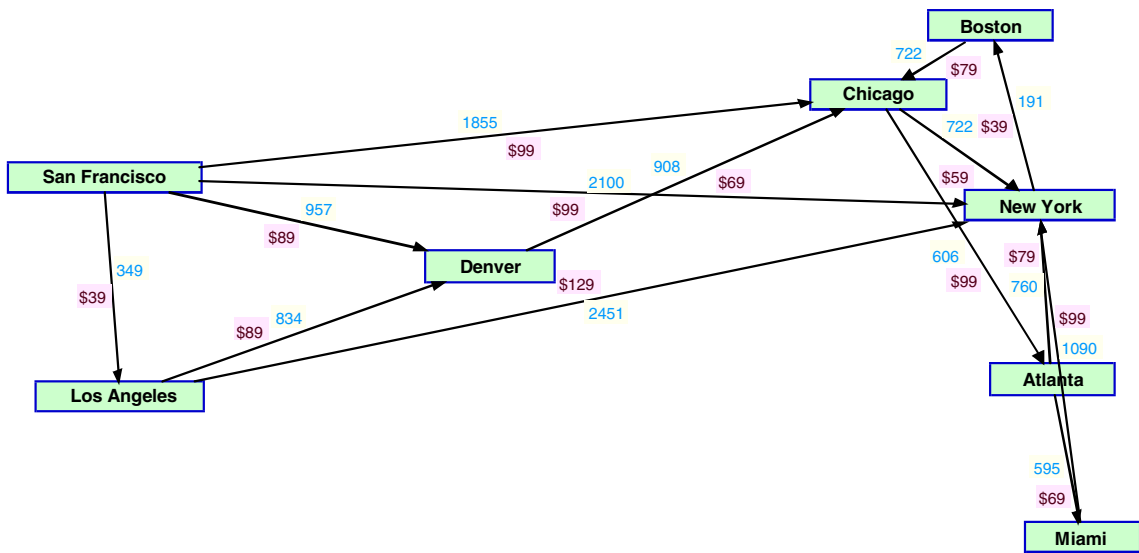


Fig. 1. A US map with airline routes where labels are used to convey fare and distance information.

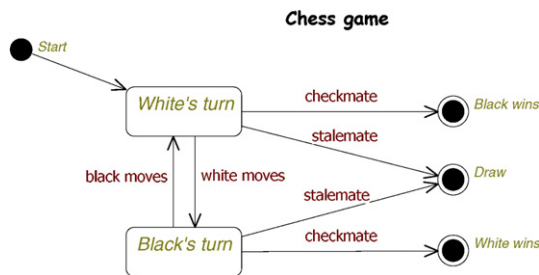


Fig. 2. A UML state diagram that makes use of labels for states and transitions as well as the title of the diagram.

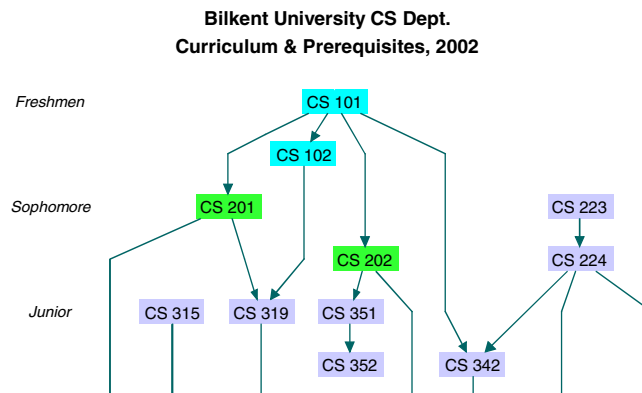


Fig. 3. A portion of a diagram for visualizing course dependencies in a department curriculum, making use of graph labels.

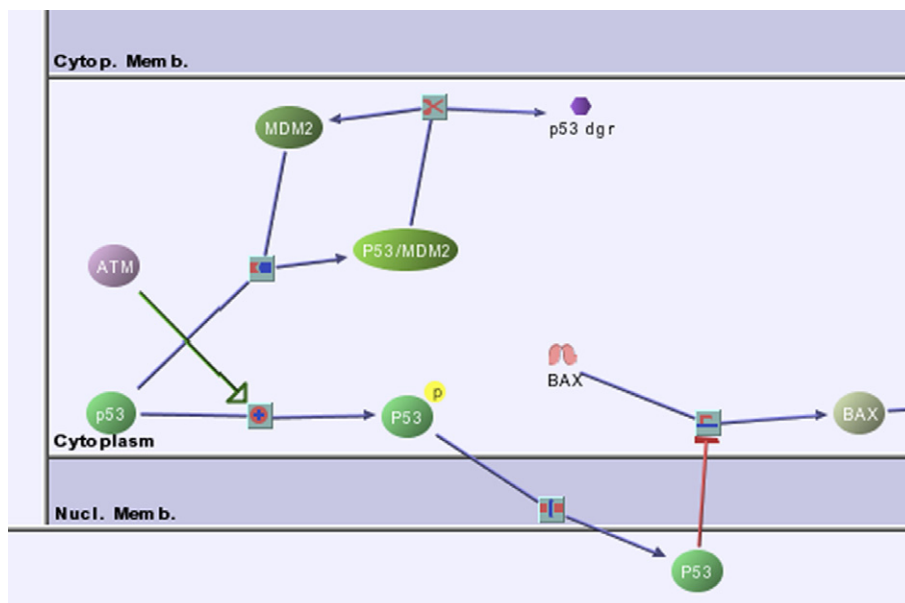


Fig. 4. Part of a pathway diagram for visualizing molecule interactions in a cell, making use of node and graph labels.

In the next section we discuss the objectives of labeling graph objects. Then we describe an appropriate framework for integrating labels into graphs; thus labels can be easily modeled, drawn, and edited within a graph visualization tool. In the following section we review available algorithms for automatic label placement as well as our extensions for integration into this graph visualization tool. Finally we detail out the labeling framework as implemented within the Graph Editor Toolkit and conclude with some remarks.

2. Labeling objectives

Labeling of edges, nodes, and graphs themselves aims to communicate the attributes of these graph objects in the most convenient way. This requires that labels be positioned in the most appropriate places.

Good label placement aids in conveying information and enhances the aesthetics of the drawing. It is difficult to quantify all the characteristics of good label placement since human visual perception, intuition, tradition and/or experience are all involved. However, one can follow some basic rules:

- *Elimination of ambiguity:* A label which is associated with exactly one graph object, must not overlap any other object in the drawing.

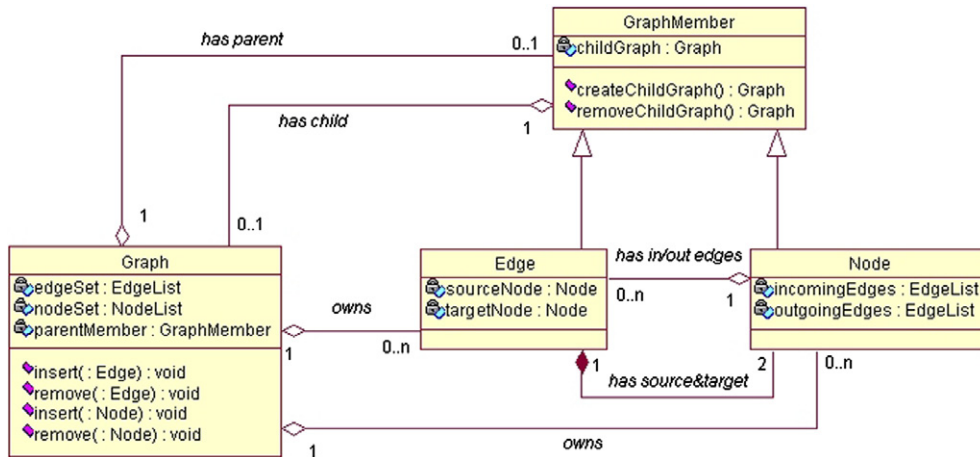


Fig. 5. A UML class diagram with labels representing roles and multiplicity of the class associations.

- *Clarity*: The relationship between labels and the associated graph object (edge, node or graph) should be easily identified without cluttering the drawing. Labels should be positioned close to, but not overlapping the associated graph object if possible. Graph labels should be positioned to label the entire drawing rather than being associated with a node or edge.
- *Flexibility*: However placement constraints on the labels should be allowed. For instance, some applications require that an edge label be associated with one of the endpoints, or the middle, of an edge (Fig. 5).

It is important to emphasize here that the user must be able to customize the rules of label placement quality to meet specific needs. For example, the user must be able to specify that the preferred position for an edge label is closer to the source or target node of the associated edge. Similarly, the user might prefer a node label to be drawn immediately below the node when possible.

Building a framework that supports automatic labeling presents two main challenges:

- Devising efficient algorithms that produce high quality label placement compared with manual placement.
- Building a labeling interface that is flexible enough to accommodate specific requests for good placement for a variety of applications.

Our framework and algorithms aim to meet these goals.

3. Framework for drawing labels

Let us first give some definitions. The *bounding rectangle* or *bounds* of a node is the tightest rectangle that contains the node's UI (i.e., graphical representation), not taking the bounds of any labels into account. The *bounds* of a label is defined similarly. The *bounding rectangle* or *bounds* of a graph is the tightest rectangle that contains the bounds of all its nodes, edges, and their associated labels (but excluding the bounds of labels associated with the graph itself).

Each label is represented by a rectangle in our framework. A label's position is determined by a triple: an *attachment point*, a *reference point*, and an *offset value* from the attachment point to the reference point. The attachment point is where the label "attaches" to the associated graph object. The reference point is a point on the label, linking the label to this attachment point on the owner object with a specified offset.

The attachment point of a node or graph label is given with respect to the node and graph bounds, respectively. The attachment point of an edge label, on the other hand, is given as a percentage distance from the edge source based on the total edge length. The reference point of a label is given with respect to the bounds of the label. The actual location of a label is calculated based on its current values for these three parameters (Fig. 6).

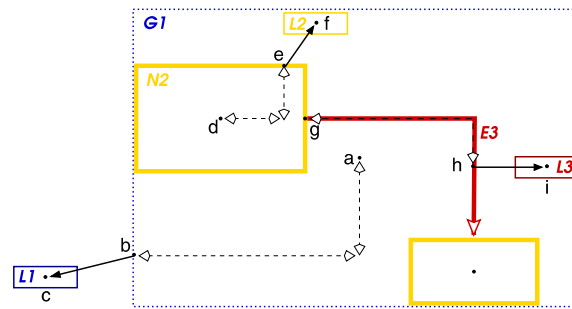


Fig. 6. Determining a label's position involves calculating the values of three parameters: attachment point (b, e, h), reference point (c, f, i), and offset (bc, ef, hi). The labels belong to three different graph objects with center or source points (a, d, g).

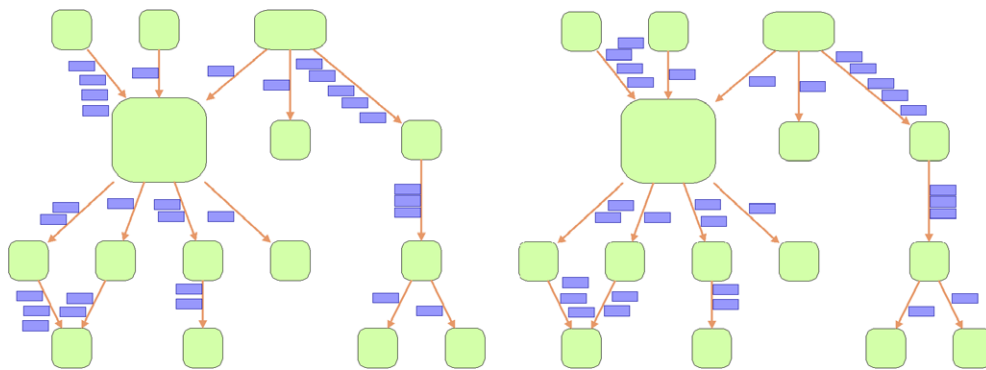


Fig. 7. Label style can be set such that the label is placed to the left or right of its owner edge.

3.1. User constraints and tailoring options

The Graph Editor Toolkit's tailoring options for labeling are quite flexible and allow the users to customize the system to their specific needs. It is important to emphasize that a user must be able to customize the rules of label quality to meet specific needs and/or expectations. Depending on the particular application, usually a different set of rules govern the preferred label positions. A user can specify the preferred position to place a label with respect to the associated graph object by specifying a distinct *style*, *association*, and *orientation* as defined below:

- *Style*: The style of a label specifies whether the label should be placed *above* or *below* (e.g., for horizontal or nearly horizontal edges), or to the *left* or *right* (e.g., for vertical or nearly vertical edges) of the associated graph object. Fig. 7 shows examples.
- *Association*: The association of a label specifies whether the label should be placed towards the *top*, *center*, or *bottom* of its owner graph object. For instance, this option can attach an edge label to the source, center, or target of the edge, respectively. See Fig. 8 for an example.
- *Orientation*: The style and association of a label might have different meanings depending upon the orientation chosen. One can use either a *global* orientation of the drawing (based on the y-coordinates of the endpoints of the edge), or an *edge* orientation (based on the direction of each edge).

4. Interactive operations

The GET labeling engine supports a number of interactive operations which are useful, especially when a user wants to manually interact with a drawing.

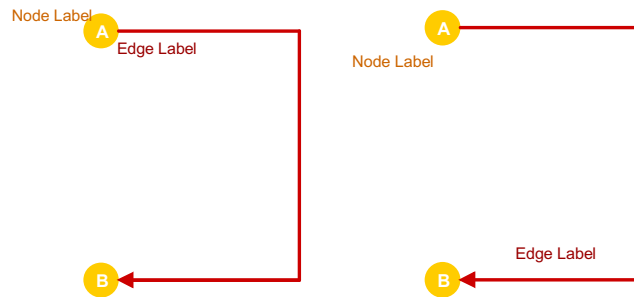


Fig. 10. The user might decide to change the position of a label with respect to the geometry of the owner graph object (e.g., move the edge label towards the target node and the node label towards the lower-left of the node).



Fig. 11. A label is tailored to be to the left of the edge, following the direction of edge (from source to target node). The label switches sides as the target node is moved to be above the source. Notice how the label is still to the left of the edge when following the edge direction.

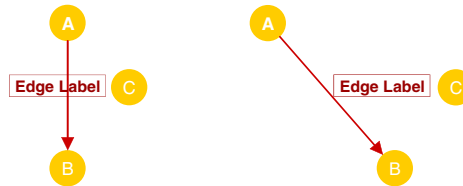


Fig. 12. Label-edge overlap is resolved when the edge moves away from a neighboring node to make room for the label.

If the user directly performs an interactive operation on a label, the values of these parameters need to be recomputed. For instance, suppose the user drags an edge label by mouse. First, the point on the owner edge that is closest to the label's new position is calculated and becomes the new attachment point. The new value of the offset is set to be the vectorial difference between the attachment and reference points. Fig. 10 illustrates this with an example.

In addition, special care has been given to edge labels with respect to interactive operations on the owner edges. When we reroute edges, their corresponding labels follow the owner edges, during which they should retain their tailoring options as well. Thus, in many cases we have to adjust the label position, after its owner edge has been moved, in order to respect the tailoring options (Fig. 11). In some cases during this interactive operation we improve the position of edge labels whenever possible. For instance, suppose a label intersects its corresponding edge. When the edge moves away from the label, the label adjusts so that it barely touches the edge (Fig. 12).

5. Automatic label placement

The automatic placement of text or symbol labels corresponding to graphical objects is critical not only in Graph Drawing but also in several other areas including Cartography [3] and Geographical Information Systems [11]. In map production, labeling is typically partitioned into three tasks: labeling points (e.g., cities), labeling lines (e.g., roads or rivers), and labeling areas (e.g., lakes or oceans). A similar terminology is used

in labeling of graphs. The problem of assigning labels to a set of points or nodes, called *Node Label Placement* (NLP), has been studied widely [3]. However the problem of assigning labels to a set of lines or edges, known as *Edge Label Placement* (ELP) has not been as well studied until recently [16,15,24,27]. Both the NLP [9,19,22] and ELP [16] problems are known to be NP-Hard. For the general labeling problem (i.e., the labeling of nodes, edges, and graphs), most labeling algorithms are based on local and exhaustive search algorithms [6,8,10]. An algorithm that transforms labeling to a matching problem is presented [17].

For graphs, automatic label placement can be done either as the graph layout is done or as a post-layout operation (i.e., performed on a fixed geometry of nodes and edges). The former approach is obviously less restricted and has the potential to yield better results, but because every type of graph or layout style might require a different method of integrating labeling into layout, it is not widely studied [1,2,21]. The latter approach, however, has received much attention over the past decade.

The implementation of the GET labeling engine is based on the algorithms presented in [15,18,17]. We have made a number of improvements, extensions and adjustments to these algorithms to integrate them into the professional visualization tool. Please see Fig. 13 for a sample drawing with labels in GET.

5.1. Basic labeling algorithms

The placement of labels in GET is a post-layout operation (i.e., performed on a fixed geometry of nodes and edges).

Our labeling technique follows three basic steps: A set of discrete potential label solutions for each object is carefully selected; this set of labels is reduced by removing heavily overlapping labels; finally, an assignment of labels to available label solutions is performed by solving a variant of the matching problem.

Basic Labeling Algorithm

INPUT: A drawing Γ and a set F of objects to be labeled.

OUTPUT: A label assignment free of overlaps.

1. Select label positions for each object in F .
2. Remove heavily overlapping labels and group overlapping labels together.

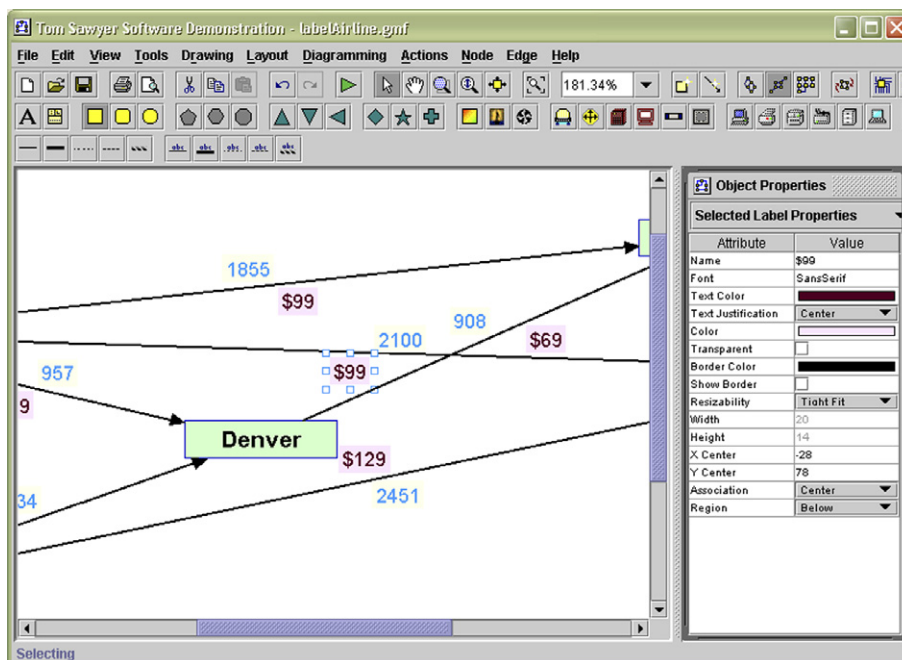


Fig. 13. A sample graph drawing with labels in GET.

3. Make the final label assignment by solving a matching problem, where at most one label position from each group is part of the solution.

When drawings are very dense or there is a large number of oversized labels, the above algorithm produces a label assignment which might be unsatisfactory (i.e., some objects might not have labels assigned to them); in such cases, a post-processing step is necessary (see Section 5.3).

5.1.1. Selecting labels

We use a number of heuristics to find a set of discrete potential label solutions for each object.

For graph labels, we define a number of label positions outside the graph bounds (i.e., inside margins). These label positions do not intersect each other and touch the bounding rectangle of the graph.

For node labels, we first define the solution space (inside or outside the node bounds) according to user preference. The solution space is the area where labels can be placed. Then we use techniques that find candidate labels within a polygon as described in [24].

For edge labels, we define a number of label positions taking into account the topology of the drawings. Most labels are parallel to the horizontal axis, and have approximately equal height; thus we have developed a technique for selecting label positions for graphs with non-horizontal edges. The goal of this technique is to assign to each object a label position that is free of overlaps and touches only its associated object. The main idea of our technique is the following:

We produce groups of label positions such that each label position in a group overlaps every other label position in the same group. Thus we produce mutually disjoint sets of label positions. We skip the next step of the Basic Labeling Algorithm, which is to reduce the set of potential label positions, since the set of potential label positions is already reduced.

We create the initial set of label positions in the following way. We divide the input drawing into consecutive horizontal strips of equal height. The height of each strip is equal to the (maximum) height of the labels. Next, we find a set of label positions \mathcal{A}_e for each edge e . Each label position must be inside a horizontal strip. We slide labels until a label touches its edge, say e . We include that label position into set \mathcal{A}_e , if it does not overlap any other graph object, or only overlaps label positions of some edge other than e . We do not allow label positions that overlap nodes or edges of the layout nor can label positions intersect with the associated edges. Label positions lie entirely inside horizontal strips. Thus, label positions can only overlap other labels in the same horizontal strip. Hence, the following always hold:

- A label position for an edge e does not overlap any other label position for e .
- If two label positions overlap, they are inside the same horizontal strip.
- Each label position overlaps at most one other label position [15].

If two label positions overlap then they belong to the same group. If a label position is free of overlaps then it belongs to a single member group.

The size of the initial set of label positions must be kept reasonably small since it affects the performance of any labeling algorithm.

The above method of defining a set of potential label positions is very practical and effective because it partitions the solution space and identifies the areas of the drawing where conflicts of label assignment may occur. In addition, it significantly reduces the search space for potential conflicts (overlaps).

5.1.2. Reducing labels

In order to reduce the set of label positions, we first create an intersection graph, where each label position is a node; whenever two label positions intersect, there is an edge connecting their corresponding nodes. Then we remove heavily overlapping labels and group the remaining label positions. Our goal is to group the label positions in such a way that overlapping labels belong to the same group. In the third step, we produce a label assignment free of overlaps by selecting at most one label from each group as part of the solution.

The optimal solution would have the maximum number of minimum size complete subgraphs (groups) of the intersection graph, with the additional constraint that each object would have a large number of label

positions as part of some groups. A successful label assignment is most likely when each object has a large number of potential label positions associated with it. In reality we want to find an independent set of complete subgraphs of label positions. Here we can use heuristics based on techniques that solve the independent set problem.

In principal we remove heavily overlapping labels, while maintaining a large number of potential labels for each object f by keeping track of the number of labels associated with f . In the end we want to reduce the intersection graph into a set of disconnected subgraphs.

At the outset, we can make this process more efficient by applying a preprocessing step that eliminates unnecessary label positions or assigns labels in obvious cases. Next we remove an appropriate number of potential labels until each of the remaining labels belongs to some group.

In the preprocessing step we heuristically remove label positions that add to the complexity of the problem without potentially improving the solution. For example, if a label position l of an object f is free of overlaps, then we can safely remove all label positions for f with lower ranking than l . We have developed a simple but successful (according to our experiments) technique for removing overlapping labels: if a subgraph C must be split, then we remove from C the node with the highest degree, unless that node corresponds to a label position of some object with very few label positions. In that case we remove the next highest degree node from C . We repeat this process until C either splits into at least two disjoint subgraphs, or is complete.

5.1.3. Matching labels to objects

In the next step, objects are matched to label positions by allowing at most one label position from each group to be part of a label assignment. Once we have the groups of overlapping labels we construct a bipartite *matching* graph. Each group of overlapping labels and each object to be labeled is a node in the matching graph. For each label position l of graph object f that is a part of a group g , there is an edge connecting the node (of the matching graph) associated with g to the node (of the matching graph) associated with f .

A maximum cardinality matching of the matching graph assigns labels to a maximum number of objects. Because the label assignment is free of overlaps, the cost of each label position will depend only on the ranking of that label. The ranking of each label position depends on user preferences. This implies that the cost of each label position can be computed by a preprocessing step. The cost of assigning a label to an object is the weight of the edge that connects the group (to which the label belongs) and the node that corresponds to the object. Therefore, a maximum cardinality, minimum weight matching for the matching graph produces a label assignment of minimum cost (among all assignments) such that at most one label position is taken from each group. Hence the following theorem:

Theorem 1. *Let A be the set of label positions for all objects of a drawing, and let A be partitioned into groups such that any two label positions of A overlap if and only if they belong to the same group. Then a maximum cardinality, minimum weight matching of the corresponding matching graph produces an optimal solution, for the set of label positions A , with no overlaps.*

The size of the matching graph depends not only on the size of the input drawing, but also on the number of label positions and number of overlaps between label positions. Unfortunately the number of label positions can be large for the size of the original graph. This implies that a typical matching algorithm might take a long time. The best algorithms for finding a maximum cardinality minimum weight matching of a graph take more than quadratic time for the size of the graph [12,23].

For the technique we have described above for non-horizontal edges, we have constructed a heuristic that finds a maximum cardinality matching with low total weight in linear time with respect to the size of the graph, by taking advantage of the structure and properties of the matching graph.

This fast heuristic takes advantage of the simple structure of the matching graph as follows. By construction, each node corresponding to a group has degree at most 2. The following algorithm finds a maximum cardinality matching for the matching graph $G_m(V_o, V_g, E)$, where each node in V_o corresponds to an object to be labeled and each node in V_g corresponds to a group of overlapping labels.

Algorithm Fast Matching

INPUT: Matching graph $G_m(V_o, V_g, E)$.

OUTPUT: A maximum cardinality matching for G_m with low total weight.

1. If the minimum weight incident edge of a node in V_o connects this node to a node in V_g of degree 1 then:
 - 1.1. Assign this edge as a matched edge.
 - 1.2. Update G_m .
2. If a node in V_o has degree 1 then:
 - 2.1. Assign its incident edge as a matched edge.
 - 2.2. Update graph G_m .
3. Repeat steps 1 and 2 until no new edge can be matched.
4. Delete all nodes of degree 0 from G_m .
5. For each node o in V_o do
 - 5.1. Remove all but the two incident edges of o with the least weight.
6. The remaining graph consists of simple cycles and/or paths.
 - 6.1. For each cycle or path find at most two maximum cardinality matchings.
 - 6.2. Choose the matching of minimum weight.

Theorem 2 [15]. *The matching produced by Algorithm Fast Matching is a maximum cardinality matching.*

Theorem 3 [15]. *Let Λ be the set of label positions for all graphical features. If every label position in Λ overlaps at most one other label position in Λ , and each graph object has at least two label positions, then Algorithm Fast Matching produces a label assignment in linear time, where at least half of the graph objects are assigned their best label position, and the rest of the graph objects their second best label position.*

5.2. Multiple labels

The algorithms have been extended to support placement of more than one label per graph object. Multiple labels per graph object are needed not only when graph objects are very long (e.g., long edges) and repetition is necessary, but also when more than one attribute per graph object must be displayed (Fig. 1).

An iterative approach has been applied to solve the problem of assigning multiple labels to each graph object of a drawing. At each iteration, one label is assigned to each graph object of the drawing. Each successive round respects the previously placed labels and reduces the solution space accordingly. This method consists of a main loop, and we execute the loop as many times as the number of labels per object. In particular, at the i th execution of the loop, we assign the i th label to each object. An algorithmic scheme of this technique can be realized in the following way:

Algorithm for Multiple Labels

INPUT: A graph Γ , a set of graph objects F in Γ to be labeled, a number N of labels for each graph object f in F .

OUTPUT: A label assignment.

1. Find an initial set of label positions L .
2. For $i = 1$ to N do:
 - 2.1. Assign the i th label to each graph object in F from the set L of potential labels using the above algorithms.
 - 2.2. Remove potential label positions from L that overlap already assigned labels.

5.3. Further improvements

The technique for non-horizontal edges produces a set of potential label positions; it splits the drawing into horizontal strips and then assigns labels to those edges that intersect the strips. It is clear that the longer and the more vertical the edge, the more potential label positions are associated with it, and the greater the

possibility that the labeling algorithm will assign a label to it. Hierarchical drawings are particularly suitable for this algorithm since edges are usually long and almost vertical (in fact, our technique was originally devised for automatic label placement in hierarchical drawings). If a hierarchical drawing is drawn from left to right, we rotate the drawing 90° and apply our labeling technique.

After extensive experimentation [15], we noticed that this technique also performs well for straight-line drawings, such as those produced by force-directed and circular techniques. However one weakness of our labeling technique is that it ignores horizontal edges or edge segments, and is therefore not suitable for orthogonal drawings. However, one can use the general technique by dividing an orthogonal drawing into horizontal and vertical strips in order to find a set of label positions, followed by the assignment of labels to edges. Fig. 14 shows the results with an example.

At last post-processing step is performed if necessary. Labels are assigned to objects by locally shifting already assigned labels followed by a limited number of backtracking operations.

The algorithm tries to place labels to respect the tailoring options (preferred position) as discussed earlier. If unsuccessful it tries to find a place that is as close to the preferred position as possible (acceptable position). In the final label assignment produced by the algorithm, no label will overlap other labels or nodes or edges other than its associated object.

In the case of the technique for non-horizontal edges, we consider shifting labels not inside horizontal strips, but rather vertically and/or horizontally in such a way that they intersect a line defining the border of some horizontal strip. It is important to remember that edges that are almost horizontal have a few label positions associated with them during the execution of the algorithm; the post-processing step must pay extra attention to such edges.

In addition, there are a number of global tailoring options that can be used to fine-tune the algorithm. When drawings are very dense or there is a large number of oversized labels, the default label assignment produced by the labeling system might not be satisfactory. In such cases, the user can fine-tune the algorithm either by requiring the labeling algorithm to spend more time in the post-processing step as discussed later, or by relaxing the labeling quality constraints to allow overlaps. For the former case, a *label positioning quality* parameter ranging from 1 to 10 sets the *intensity* of the algorithm, which is mostly related to the post-processing step. A higher integer value results in more accurate positioning of labels under normal circumstances but it takes longer to execute. For the latter case, the *allowed overlap percentage* parameter determines if the labels are allowed to overlap with one another and with other graph objects. When set to 10%, for example, each label's dimensions are treated as if they were 10% smaller; this increases the overall success of the algorithm at a cost of up to 10% overlap. Fig. 15 illustrates this tailoring option with an example.

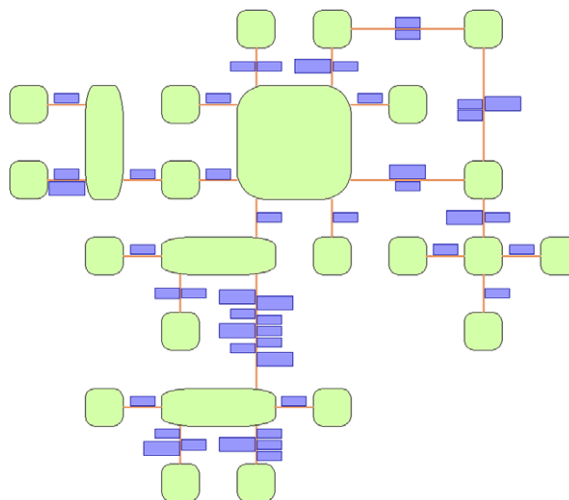


Fig. 14. Edge labels in an orthogonal drawing with many horizontal edge segments.

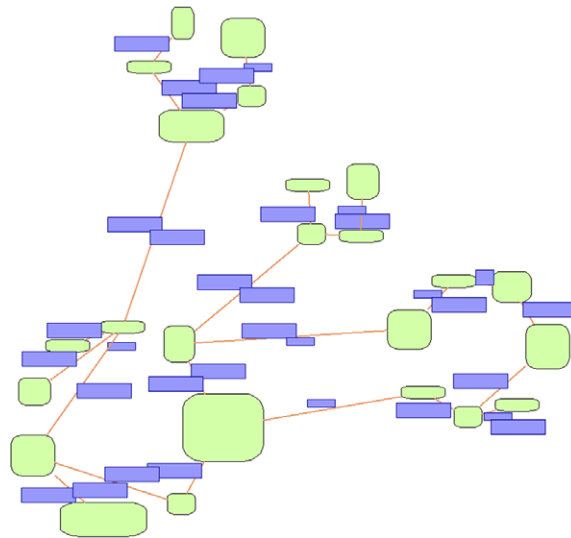


Fig. 15. A circular drawing where labels are allowed to overlap (by as much as a specified percentage) other graph objects.

6. Conclusion

We have presented a framework for modeling, drawing, editing, and automatic placement of labels in a graph. The framework has been integrated into a graph visualization tool, namely the Graph Editor Toolkit; existing automatic label placement algorithms have been extended and improved. The interface to these algorithms comes with per label and per graph tailoring options that not only provide input to the label placement algorithms about specific constraints on the position of labels but also to adjust several parameters that let the user fine-tune these algorithms.

One essential future research goal on labeling is the design of efficient interactive and incremental labeling algorithms for dynamically changing graphs.

References

- [1] C. Binucci, W. Didimo, G. Liotta, M. Nonato, Labeling heuristics for orthogonal drawings, in: P. Mutzel, M. Jünger, S. Leipert (Eds.), *Graph Drawing (Proc. GD '01)*, Lecture Notes in Computer Science, vol. 2265, Springer-Verlag, 2002, pp. 139–153.
- [2] C. Binucci, W. Didimo, G. Liotta, M. Nonato, Orthogonal drawings of graphs with vertex and edge labels, *Comput. Geom.* 32 (2) (2005) 71–114.
- [3] J. Christensen, J. Marks, S. Shieber, An empirical study of algorithms for point feature label placement, *ACM Transactions on Graphics* 14 (3) (1995) 203–232.
- [4] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, *Graph Drawing, Algorithms for the Visualization of Graphs*, Prentice-Hall, Upper Saddle River, NJ, 1999.
- [5] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, *Comput. Geom. Theory Appl.* 4 (1994) 235–282.
- [6] J.S. Doerschler, H. Freeman, A rule based system for dense map name placement, *Commun. ACM* 35 (1) (1992) 68–79.
- [7] U. Dogrusoz, Q. Feng, B. Madden, M. Doorley, A. Frick, Graph visualization toolkits, *IEEE Comput. Graph. Appl.* 22 (1) (2002) 30–37.
- [8] L.R. Ebinger, A.M. Goulete, Noninteractive automated names placement for the 1990 decennial census, *Cartogr. Geogr. Inform. Syst.* 17 (1) (1990) 69–78.
- [9] M. Formann, F. Wagner, A packing problem with applications to lettering of maps, in: *Proc. 7th Annual ACM Sympos. Comput. Geom.* 1991, pp. 281–288.
- [10] H. Freeman, J. Ahn, On the problem of placing names in a geographical map, *Int. J. Pattern Recognit. Artif. Intell.* 1 (1) (1987) 121–140.
- [11] H. Freeman, Computer name placement, in: D. Maguire, M. Goodchild, D. Rhind (Eds.), *Geographical Information Systems: Principles and Applications*, Longman, 1991, pp. 445–456.
- [12] A. Goldberg, R. Kennedy, An efficient cost scaling algorithm for the assignment problem, *Math. Progr.* 71 (1995) 153–178.

- [13] Graph Editor Toolkit User's Guide and Reference Manual, Tom Sawyer Software, Oakland, CA, USA, <http://www.tomsawyer.com> (1992–2006).
- [14] JViews User's Guide, 94253 Gentilly Cedex, France, <http://www.ilog.com> (2006).
- [15] K. Kakoulis, I. Tollis, An algorithm for labeling edges of hierarchical drawings, in: G. Di Battista (Ed.), *Graph Drawing (Proc. GD '97)*, Lecture Notes in Computer Science, vol. 1353, Springer-Verlag, 1998, pp. 169–180.
- [16] K.G. Kakoulis, I.G. Tollis, On the complexity of the edge label placement problem, *Comput. Geom. Theory Appl.* 18 (2001) 1–17.
- [17] K.G. Kakoulis, I.G. Tollis, A unified approach to automatic label placement, *Int. J. Comput. Geom. Appl.* 13 (1) (2003) 23–60.
- [18] K.G. Kakoulis, I.G. Tollis, Algorithms for the multiple label placement problem, *Comput. Geom. Theory Appl.* 35 (3) (2006) 143–161.
- [19] T. Kato, H. Imai, The NP-completeness of the character placement problem of 2 or 3 degrees of freedom, in: *Record of Joint Conf. of Electrical and Electronic Engineers in Kyushu*, 1988, pp. 11–18, in Japanese.
- [20] M. Kaufmann, D. Wagner (Eds.), *Drawing Graphs: Methods and Models*, Lecture Notes in Computer Science, vol. 2025, Springer, 2001.
- [21] G.W. Klau, P. Mutzel, Combining graph labeling and compaction, in: J. Kratochvíl (Ed.), *Graph Drawing (Proc. GD '99)*, Lecture Notes in Computer Science, vol. 1731, Springer-Verlag, 1999, pp. 27–37.
- [22] J. Marks, S. Shieber, The computational complexity of cartographic label placement, Tech. Rep. 05–91, Harvard University, 1991.
- [23] R. Tarjan, Data structures and network algorithms, in: *CBMS-NSF Regional Conference Series in Applied Mathematics*, vol. 44, 1983.
- [24] J.W. van Roessel, An algorithm for locating candidate labeling boxes within a polygon, *Am. Cartogr.* 16 (3) (1989) 201–209.
- [25] A. Wolff, T. Strijk, The map-labeling bibliography: www.math-inf.uni-greifswald.de/map-labeling/bibliography, 2003.
- [26] yFiles User's Guide, yWorks GmbH, D-72076 Tübingen, Germany, <http://www.yworks.com>, 2006.
- [27] S. Zoraster, The solution of large 0–1 integer programming problems encountered in automated cartography, *Operat. Res.* 38 (5) (1990) 752–759.