User Hints for Directed Graph Drawing

Hugo A.D. do Nascimento* and Peter Eades**

Basser Department of Computer Science,
The University of Sydney, Australia
{hadn,peter}@cs.usyd.edu.au

Abstract. This paper investigates an interactive approach where users can help a system to produce nice drawings of directed graphs by giving hints to graph drawing algorithms. Hints can be three kinds of operations: focus on a specific part of the drawing that needs improvement, insertion of layout constraints, and manual changes of the drawing. These hints help the system to escape from local minima, reduce the size of the solution space to be explored, and input domain knowledge. The overall aim is to produce high quality drawings. We present a system based on this approach and a pilot study involving human tests.

1 Introduction

Many graph drawing methods have been developed to produce drawings of graphs to satisfy aesthetic criteria, such as showing few edge crossings and monotone edge direction (eg. upward or downward). The optimization problems inherent in graph drawing are mostly NP-hard. The aesthetics may also conflict, that is, there is no optimum solution for two criteria simultaneously. As a consequence, graph drawing methods are mainly heuristics that work reasonably fast, but may result in poor quality drawings. Even amongst papers in Graph Drawing [1,2,3,4], one can find drawings that are produced in a few seconds, but present many edge crossings, edge bends, no symmetry, etc. In application areas such as Software Engineering, the drawings are often worse; see [5]. For instance, drawings of the Unix System Family tree that appear in many papers about directed graph drawings show at least one edge crossing [2]. It is interesting to note that this graph is upward planar.

There are several approaches for dealing with the weakness of automatic graph drawing methods. The most popular one is to apply the method for generating an initial drawing, and then improve the drawing manually. This is perhaps the most common way of creating a winning drawing for the *Graph Drawing Contest* [6]. The automatic method solves a great part of the problem by finding approximate positions for the vertices, and then the user produces the final layout by attending to other problems (including problems driven by domain knowledge) that the heuristics do not solve. In many cases, the user can easily

* Lecturer of the Inst. of Informatics, UFG-Brazil. PhD scholarship of CAPES-Brazil.
** Supported by the Australian Research Council.

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 205–219, 2002.

recognize part of the drawing that needs to be improved and a way of improving it, as long as a good initial drawing is provided.

An alternative approach is to develop better (and more complex) algorithms that consider several rules about how to draw edges and vertices, or to use metaheuristics that explore the space of possible layout solutions. It is not clear, however, that better algorithms and methods can ever eliminate the possibility of having the user doing some post-processing. This is because there are always some instances for which sufficiently fast algorithms do not produce the best solution, or the space of solutions is too large to be adequately searched in a reasonable amount of time. Moreover, it is common to have a large number of good equivalent layouts for the same graph, where the decision of which one to be taken is subjective, or domain dependent. Even when some subjective aspects can be modeled as objective functions and constraints in flexible algorithms [7,8,9], it is difficult to ensure that all user preferences are considered, and that they imply no ambiguity by leading to a single "optimum" solution. It may also be difficult to assure that the algorithm will find this solution. Thus, in the most extreme situation, the user is still important for validating the result produced by automatic methods or for selecting between a number of good drawings.

In this paper we investigate an approach for drawing directed graphs that considers the user as a *fundamental* element of the graph drawing process. The user plays a very important role by cooperating with an optimization method to produce good quality drawings. Basically, the user provides *hints* to an optimization method, helping it to escape from local minima, to converge much faster to better results, or to clarify subjective and domain dependent aspects.

We apply our user-hints based approach to the problem of drawing directed graphs resulting in an interactive system. Directed graphs were chosen since they appear in several real applications and involve many difficult graph drawing problems. We use the popular method of Sugiyama et al. [2] as our main optimization method. A human pilot study of our system is done in order to analyze the contribution of user hints to the quality of graph drawings. This paper is organized as follows. Section 2 describes some related research regarding the contribution of users to optimization processes. Section 3 presents our approach based on user hints, and Section 4 describes how hints can be incorporated to the Sugiyama method. An interactive system that supports hints is presented in Section 5. Section 6 discusses our experiments with the system and the results obtained. Finally, Section 7 draws our conclusions about interactive graph drawing based on user hints and offers future research directions.

2 Related Work

Due to the high complexity of optimization problems and the constant demand for solutions close to the optimum, some new research has been done in the direction of allowing users to cooperate with automated optimization methods.

A good example of such studies is the work presented by Anderson et al in [10], where the authors introduce a cooperative paradigm called *Human-Guided*

Simple Search (or *HuGSS* for short). HuGSS divides an optimization process into two main subtasks carried out by different entities: the computer is responsible only for finding local minima using a simple hill-climb search; the user is responsible for escaping from the minima leading the search to better solutions. Some visualization techniques help to identify parts of a solution that are promising for improvement. Then the user can either manually change the solution or focus the search on the identified parts. The focus consists of setting a high search priority for a part of the solution and/or defining how deep the search will be executed. The HuGSS paradigm was applied to the problem of capacitated vehicle routing with time windows, and it showed that a simple hill-climb algorithm could be significantly enhanced by user interaction. The same idea was used for the graph clustering problem [11].

Another interesting work on user interaction appears in [12]. It consists of an interactive constraint-based system for graph drawing using force-directed placement [13]. The system works as follows: a graph is modeled as an energy system with springs linking all pairs of vertices, and a method for quadratic optimization is set to continuously compute a layout that correspond to a state of minimal energy. While the optimization method is running, the user can incrementally add constraints to the model, in order to confirm the drawing to his or her desires. Constraints are called VOFs (*Visual Organization Features*) and include a variety of layout aspects such as: showing two vertices close to each other, showing an edge as an orthogonal line, etc. VOFs are implemented as extra springs and added to the original energy model. The system solves constraints searching for a new state that minimizes the energy of the entire set of springs (composed by the original springs of the graph plus the constraint springs). The user may manually move some vertices to help the optimization method to escape from local minima. This characterizes a very general and flexible approach for solving layout constraints that are added during run time.

User guidance can also occur in a much softer way, not by setting algorithms to do specific tasks, but indicating whether or not they are on the right path. This approach was adopted by Branke et al [14,15] using Genetic Algorithm (GA) for drawing general graphs. In their system a genetic algorithm tries to minimize a weighted function of seven aesthetic criteria (minimum number of crossings, high angular resolutions, many symmetries, etc.) as it is commonly done by other meta-heuristics for Graph Drawing [7,8]. However, instead of executing the meta-heuristic for a long period of time until the drawing has converged, the system stops every few iterations and gets feedback from the user. Basically, it shows the best eight drawings currently produced (one drawing for each one of the seven aesthetic criteria, and the best drawing according to the weighted function), and asks the user for scores between 0 to 9 for each drawing. Then the system uses those scores to adjust the weight of each aesthetic criterion in the weighted function. For instance, giving high scores to drawings with few edges crossings contributes to an increasing of the relative weight assigned to the edge crossing criterion in the system. At the end, the system "knows" the relative

importance of each criterion that describes the user's aesthetics for a specific drawing.

All these pieces of research show, in different ways, an emerging trend for having users guiding optimization processes. Our approach is close to the HuGSS paradigm in the sense that users are more active and provide pieces of information directly to the optimization method. However, we also consider the possibility of adding constraints to the problem in runtime and use other optimization methods than just hill-climbing ones. The next sections describe our approach in details.

3 Hints for Directed Graph Drawing

We use *user hints*, or just *hints*, to refer to the information provided by users to optimization methods. A hint should help the system to escape from local minima, to accelerate the optimization process, or to solve ambiguity in cases where there is more than one feasible solution. In the context of this paper, hints help graph drawing algorithms to search for high quality graph drawings according to a set of aesthetic criteria.

3.1 Types of User Hints

We consider three kinds of hints for directed graph drawing:

- **Focus.** The aim of focus is to reduce the space of solutions to be explored by a search method. In general, after running a graph drawing algorithm on the whole graph we get a reasonably good drawing, with some areas that do not satisfy the aesthetic criteria. By identifying this, the user can focus the drawing algorithms again on the areas with poor quality in order to improve them. This means that the algorithms will redraw only the focused areas of the graph. The layout of vertices in the non-focused areas is not changed.
- **Layout Constraints.** Layout constraints are useful for helping the system to fix bad quality aspects of a drawing, or for removing ambiguity about where to draw some vertices. We have adopted two kinds of layout constraints, *Top-Down* and *Left-Right*. The *Top-Down* constraint defines an *above-relation* between two vertices u and v, such that u has to appear somewhere above v in the drawing. Similarly, the *Left-Right* constraint defines an *on-the-left-relation* between two vertices u and v.
- **Manual Changes.** Other drawing aspects that are not easily controlled by focus and layout constraints can be fixed by manual changes. The user does manual changes only on vertices by moving them to a different position of the drawing. Changes on edges can be done by moving their related vertices. The mechanism of manual changes is already commonly used in graph drawing activities as a part of a post-processing and fine-tuning step. However, here we have a much more powerful tool, since changes in a drawing may drive the system out of a local minimum to a better solution.

3.2 A Framework for Giving User Hints

An interactive framework for giving hints is shown in Figure 1. Arrows with a capital label represent the action of giving a hint. Note that all kinds of hints are direct or indirect input to the optimization method. The drawing activity occurs as iterative steps with the user providing hints to the optimization method. Then the method produces a new drawing of the graph by taking into consideration information about the current drawing and the hints. These steps repeat until the user is happy with the drawing.

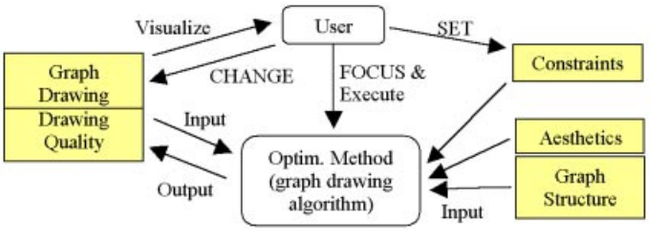


Fig. 1. Interactive framework user hints.

3.3 Implementing Focus and Constraints in the Sugiyama Method

For the purpose of drawing directed graphs, we use an optimization method based on the Sugiyama method [2]. The Sugiyama method draws a graph on a hierarchical set of horizontal lines called layers. The method consists of four steps that in general involve NP-hard problems [1,2]:

1. *Cycle Removal*: it reverses some edges of the graph in order to make it acyclic.
2. *Layer Assignment*: vertices are assigned to layers such that edges show a uniform orientation as much as possible. When an edge intersects one or more layers, it is replaced by a set of small edges (u, n_1),$(n_1, n_2), \ldots$ (n_i, n_{i+1}),(n_k, v), where n_i, $i = 1, 2 \ldots k$ are dummy vertices inserted on each intersection point, and u and v are real vertices.
3. *Crossing Reduction* : vertices are sorted in each layer in order to reduce the number of crossings.
4. *Horizontal Coordinate Assignment*: the X- coordinate of each vertex is computed making edges as straight as possible, and reducing bends and the width of the drawing. All edges changed in step 1 are also reversed to their original orientation.

We preserve the general structure of the Sugiyama method and adjust each step to support focus and layout constraints. Focus has two effects: it limits the action of the graph drawing algorithms to the focused vertices and it defines

special constraints that "freeze" the non-focused vertices. Thus, given a graph $G = (V, E)$, we focus on a selected set $A \subseteq V$, by running the Sugiyama method only on A. The X,Y coordinates of the vertices in $V - A$ are kept fixed. On the other hand, layout constraints are modeled either as extra edges added to the graph or as normal constraints that impose an ordering to vertices. Layout constraints can be defined only for real vertices. Some similar kinds of constraints for the Sugiyama method are investigated in [16] and [17].

For simplicity, in the rest of this paper we use the term *selected vertex* meaning a vertex in the selected set A, and *fixed vertex* for a vertex in $V - A$. The drawing is constructed on a grid of integer coordinates. The rows of the grid represent layers. A set of K layers is labeled $L_1, L_2, \ldots L_k$, starting from bottom to up. We use the notation l_v to indicate the layer assigned to a vertex v, with $l_v \in \{L_1, L_2, \ldots L_k\}$. The way that we implement focus and constraints is slightly different for each step of the Sugiyama Method. We explain now this implementation in details.

3.4 Cycle Removal

In the Cycle Removal step, the focus mechanism has no effect. Cycles involving selected vertices and fixed vertices are treated equally (Left-Right constraints also do not affect the drawing). Top-Down layout constraints, however, cause a great impact on the final result of this step.

As an example of how important layout constraints are, consider a graph composed of a ring with four vertices, a, b, c and d. There are four basic ways of drawing this ring such that the number of downward edges is maximum (optimal). These drawings are shown in the Figure 2(a) to (d). Without constraints, the four drawings are equivalent according to the number of downward edges. However, if the user prefers to have the vertex a drawn above the vertex c, and inserts a Top-Down constraint on a and c, this operation reduces the number of optimal solutions to only two, shown in Figures 2(a) and (b). If the user inserts another Top- Down constraint, now on b and d, this lets us with a single optimal solution, Figure 2(a). The user can go even further by inserting another Top-Down constraint to have c drawn above b. In this case, the only valid solution is to reverse the edge (b, c) and (d, a), Figure 2(e), since the constraints defines a precise order: a above c, c above b, and b above d.

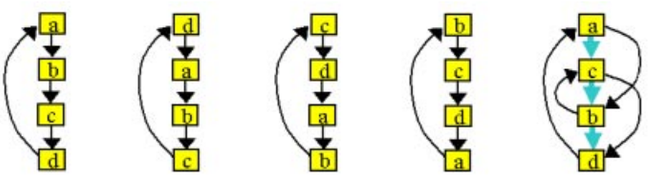


Fig. 2. Different ways of drawing a ring. Thick arrows represent constraints.

In summary, layout constraints can be used not only to reduce the number of feasible solutions, but also to force the system to considerer a specific solution. Note that all layout constraints, Top-Down and Left-Right, involve a pair of vertices. Therefore, layout constraints are modeled as special directed edges, that we call *constraint edges*. Constraint edges can be freely inserted into the system, providing that they do not make a cycle.

Considering the effect of layout constraints, we developed a new approach for the Cycle Removal step. Let $G = (V, E)$ be the graph to be drawn and L the set of Top-Down constraints. First, we construct a new graph G'' by merging L with G. Whenever an original edge and a constraint edge (excluding orientation) connect the same pair of vertices, we remove the original edge and leave the constraint. The merge procedure can be formalized as: $G'' = (V, E'')$, where $E'' = L \cup \{(u, v) \in E | (u, v) and (v, u) not in L\}$. If the resulting graph G'' is acyclic then the problem was solved. Otherwise, a method for the Feedback Arc Set problem is applied to this graph, but it is set to reverse only original edges. For instance, by merging the ring from the previous example with the set of constraints in Figure 2(e) causes the edge (b, c) to be removed. Then, the next step is to reverse some edges in G'' in order to break cycles. The algorithm for this task can reverse any edge, except (a, c), (c, b) and (b, d), which represent constraint. The optimal solution would be to reverse only (d, a).

We modify the *Greedy-Cycle-Removal* heuristic in [1] in order to solve the Feedback Arc Set problem with constraint edges; see Figure 3. The modification, highlighted in bold, is minor, however, it assures that only original edges are reversed. The advantage of using this algorithm is that it is simple and runs in linear time. The algorithm works by removing vertices from the graph and adding them either to a list S_l or to a list S_r. Finally, S_l is concatenated with S_r to form S. The list S provides a sequence of the vertices of G''. Then, all edges $(u, v) \in E'$ with v appearing before u in S are reversed, resulting in a acyclic graph.

Let G_c be a copy of G'.
1. Initialize both S_l and S_r to be empty lists.
2. while G_c is not empty do
   (a) while G_c contains a sink do
   Choose a sink u, remove it from G_c, and prepend it to S_r. (Isolated vertices are also considered sinks at this stage.)
   (b) while G_c contains a source do
   Choose a source u, remove it from G_c, and append it to S_l.
   (c) if G_c is not empty then
   Choose a vertex u, such that **there is no constraint edge *(v,u)* for any vertex *v* left in G_c**, and the difference *outdeg(u)-indeg(u)* is maximum; remove u from G_c and append it to S_l.
3. Concatenate S_l with S_r to form S.

Fig. 3. Modified version of the Greedy-Cycle-Removal heuristic presented in [1].

3.5 Layer Assignment

The Layer Assignment is executed for the graph $G'' = (V, E'')$ produced by the previous step. Focus is considered here, and it is implemented by modifying a layering algorithm for not changing the coordinates of fixed vertices. Note that edges with fixed vertices in both ends may not affect the layering algorithm, so they can be removed for saving time. This approach is presented in Figure 4.

1. Let $G'=(V,E')$ be the graph resulting from the previous step, and $A \subseteq V$ the set of selected vertices. Remove all edges (u,v) from E' with u and v fixed vertices (belonging to V-A).
2. Apply a Layering Algorithm on G'. The algorithm, however, should be modified for not changing the coordinates of the fixed vertices.

Fig. 4. Approach for Layer Assignment with constraints and focus.

We use the Longest Path Layering heuristic [18] to construct a layering of G''. This algorithm results in drawings that are in general too wide; however, it can be easily modified to handle focus, and it runs in linear time. The original version of this heuristic places all sinks in the bottom layer, L_1. Then each remain vertex u is placed in layer L_{p+1} where the longest path from u to a sink has length p. In our modified version we consider that all fixed vertices are sinks with predefined layers assigned to them, even though they may have outgoing edges. This implies that each non-sink vertex u will be placed in layer L_{p+q}, where p is the length of the longest path from u to a sink, q is the label of the layer where the sink is, and $p + q$ is maximal. Note that our layering algorithm may violate Top-Down constraints in a special case where they conflict with focus. Consider that there is a chain of directed edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$, where v_1 is a fixed vertex, $v_2 \ldots v_{k-1}$ are selected vertices, v_k is a sink (or a fixed vertex), and $l_{v_1} - l_{v_k} < (k-1)$, with l_{v_1} and l_{v_k} the layers assigned to v_1 and v_k respectively. If (v_1, v_2) is a constraint edge, then this constraint will be violated since v_2 will be assigned to a layer above v_1. All other edges (v_{i-1}, v_i), $i = 3, \ldots, k$ will point downward.

We developed a solution for the case where v_k is a normal sink. It consists of adding a post-processing step that uses the previously computed layering to shift down some vertices. Basically, for each vertex $u \in V$ taken in the topological order we assign u to a new layer $l_u = min(l_v - 1 :$ for all vertices v such that there is an edge $(v, u) \in E'')$; if there is no edge $(v, u) \in E''$ for any $v \in V$, u is kept in its current layer. The revised algorithm still runs in linear time. It moves all vertices $v_2, \ldots v_{k-1}$ $k + l_{v_k} - l_{v_1} - 1$ layers down. Unfortunately, the problem persists for the case where v_k is a fixed vertex: the vertex v_{k-1} will be assigned to a layer below vertex v_k. However, this is a problem due rather to a conflict between focus and layout constraints than to the layering algorithm itself. Our approach favors focus against layout constraints.

3.6 Crossing Reduction

In the next step of the Sugiyama method, we use the original graph $G = (V, E)$ as well as the layering defined by the previous step. A version of the barycenter algorithm [2] is applied to handle focus and Left-Right constraints. This version adjusts only the X-coordinate of selected vertices and solves constraints during the processing. A general description of the algorithm is shown in Figure 5.

Let $L_1, L_2, \ldots, L_K$ be the set of K layers defined by the Layer Assignment step.

Repeat until the number of edge crossings is minimal

1. For $i \leftarrow K$-1 to 1 do
   a. For each **selected** vertex u in layer L_i, move u to its barycenter position according to its adjacent vertices in layer L_{i+1}. If u has no adjacents in L_{i+1}, its actual position is preserved.
   b. **FixConstraints (*i*)**.
2. For $i \leftarrow 2$ to K do
   a. For each **selected** vertex u in layer L_i, move u to its barycenter position according to its adjacent vertices in layer L_{i-1}. If u has no adjacents in L_{i-1}, its actual position is preserved.
   b. **FixConstraints (*i*)**.

Fig. 5. Barycenter algorithm for Crossing Reduction with constraints and focus.

The algorithm uses a heuristic called *FixConstraints(i)* that reorganizes the vertices in layer L_i. This heuristic constructs a set S composed of all selected vertices in L_i that overlap (have the same X-coordinate), have a non-integer X-coordinate or do not satisfy one or more Left-Right constraints. Then the algorithm moves each vertex in S to an empty integer position in L_i where the number of unsatisfied constraints is minimized. If there are many equally possible empty positions in the layer, the algorithm chooses the one closest to the vertex"s current location. Left-Right constraints may involve vertices in different layers. In this case, all constraint edges that have at least one vertex sitting on layer L_i will be analyzed.

Note that our implementation of FixConstraints may not result in satisfaction of all Left-Right constraints, since it analyses locally the layers, and it demands the existence of empty positions for moving vertices. Some vertices may also be fixed, forbidding constraints to be solved. Nevertheless, we expect the heuristic to solve many constraints when applied several times in the barycenter algorithm.

3.7 Horizontal Coordinate Assignment

Finally, the last step of Sugiyama method, the Horizontal Coordinate Assignment, is not explicitly included in our approach. This is because the barycenter algorithm combined with FixConstraints already assigns X-coordinates that do

not produce many bends or long edges. Moreover, the algorithm in Figure 5 can be re-applied for improving the horizontal coordinate assignment by focusing only on vertices that cause bends or long edges.

4 The GDHints System

We implemented the Sugiyama steps described in the previous section into an interactive system, called GDHints[1]. A snapshot of the system is shown in Figure 6. The system includes:

- a user interface, by which the user can select vertices for focus, add and delete constraints or perform manual changes;
- graph drawing functions for layering (cycle removal and layer assignment) and ordering (for crossing reduction); and
- displays of quality metrics of drawings.

User-System Cooperation and Quality Feedback. The system and the user work together for improving a drawing of a graph. The drawing is improved when its new layout is better than the previous one in the following priority of aesthetic criteria: (1) number of upward edges, (2) number of edge crossings, (3) number of dummy nodes, (4) number of edge bends and (5) drawing area. At the beginning of the processing, the layering and the ordering functions are automatically executed for a new graph in order to produce an initial drawing. Then the user can call these functions again for redrawing selected parts of the graph. The system evaluates the quality of every new drawings and automatically saves the best drawing generated so far. At any time, the user can return to the best drawing or can force the system to accept the current drawing as the best one.

The system provides useful feedback for the user"s actions. This includes colors for highlighting bad quality aspects of the drawing, and sound and animation events for calling the user"s attention whenever a new solution better than the current best one is found.

5 Pilot Study

An initial study with human experiments was done for validating our approach. Five users took part in the experiments. All of them have a background in Computer Science and in Graph Drawing. The users also had a 30-minute introduction about the system, before starting the experiments.

The study involved three kinds of experiments: $E1$ (constraints only), $E2$ (constraints + focus) and $E3$ (constraints + focus + manual changes).

These experiments were done using six graphs, which details are shown in Table 1. Graphs $G3$, $G4$, $G5$ and $G6$ are from [2,3,4]. In total 90 experiments

[1] Our system can be downloaded from *www.cs.usyd.edu.au/∼visual/systems/gdhints*

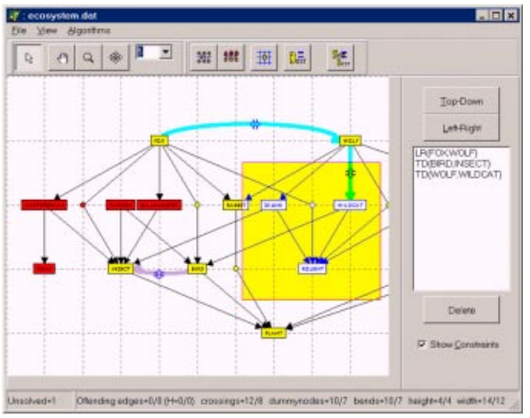


Fig. 6. The interactive system based on user hints

were done (5 users x 3 types of experiments x 6 graphs). We started with small graphs, so that the users could improve their skills in giving hints smoothly. Constraints were allowed in all experiments since they are an advanced feature in interactive graph drawing and we wanted to test them as much as possible. On the other hand, manual changes are very intuitive (the users could tend to use mainly this option). For this reason, we considered manual changes only in experiment $E3$.

The users" actions were recorded into history files for further analysis. After the experiments we also got subjective feedback from the users. The first 15 experiments (related to graph $G1$) were not included in the average analysis of the system, since we considered the users were still learning how to use the system during that time.

Table 2 shows the minimum, the maximum and the average values of the aesthetic criteria for the best drawings produced by the 5 users.

Compared to the initial solutions described in Table 1, the number of offending edges was not improved much. This is because the layering algorithm already produces a result very close to the optimum. On the other hand, there was a significant reduction of the number of edge crossings. The experiments where not all users could improve crossings were the ones based only on constraints. When focus and manual changes were allowed, all five users produced drawings with lesser crossings. Note that the number of crossings could still be smaller than the minimum presented here for some graphs, but this may result in a worse solution, with more offending edges. Regarding the numbers of dummy vertices and bends, and the area of the drawings, they were higher than the initial figures for almost all experiments. This shows that such aesthetics are in general inversely proportional to the improvement of edge crossings. Some examples of drawings produced by the users can be seen in the Appendix. Other drawings are available in our web-page.

We present relative results for experiments $E1$, $E2$ and $E3$ in Table 3. It contains, in percentage, the average values from the previous table divided by the initial values (from Table 1) and combined for graphs $G2$ to $G6$. We can see that the users could reduce the number of edge crossings by about 20% on average in experiment $E1$, 56% in experiment $E2$, and 65% in experiment $E3$. The percentages for dummy vertices, bends and area of the drawings are greater than 100% showing an increasing of the initial figures for these criteria. The values for offending edges are not considered here.

An interesting point is the gap between the experiments $E1$ and $E2$. Adding the focus facility to the system improved much the results. In fact, the users affirmed it was difficult to improve the drawings in the experiment $E1$, since adding new Left-Right constraints very often caused many edge crossings. We concluded that the system was able to find solutions that satisfy Left-Right constraints in most cases, but not the ones with minimal number of crossings.

The usage (percentage of user interactions represented by mouse clicks) of the main operations in the system is presented in Table 4. The row *Total* is the overall results for all experiments. The column *Other* includes align to grid operations, zoom in, zoom out and other actions.

Table 4 shows that constraints played a less important role in the optimization processing compared to focus (*select operations*) and manual changes (*move operations*). Ordering was the most significant operation, and it was executed for improving drawings after giving hints to the system. The users also pointed out that the option for returning to the best solution was very important. They used this facility in a simple search approach for escaping from local minima: performing changes on the drawing and executing the ordering function several times for improving the solution; after some iterations, the user presses a button to recover the new best solution found.

Table 1. Graphs used for the experiments.

Name	|V|	|E|	Quality of the initial drawings (produced by the system using the Sugiyama method)				
			Offending edges	Crossings	Dummy Vertices	Bends	Area
G1 – Waleska	10	20	2	9	17	13	72
G2 – Klayer	18	24	0	5	0	4	24
G3 – Ecosystem	15	26	0	16	7	7	48
G4 – Csyntax	34	46	4	12	39	19	182
G5 – Unix Sys. Family Tree	41	49	0	4	24	9	132
G6 - Dynworld	43	69	6	70	144	56	360

A final result obtained from the experiments regards processing time. The users spent on average 14 minutes on each experiment. However, just 10% of this time was used by the system for doing some processing. In the other 90%, that

Table 2. Quality of the best drawings produced by the users for all graphs.

Graph	Exp	Offending Edges		Crossings		Dummy Vertices		Bends		Area	
		Min-Max	Av	Min-Max	Av	Min-Max	Av	Min-Max	Av	Min-Max	Av
G1	E1	2-2	2	5-7	6	17-30	19.6	12-22	14.6	72-99	79.2
	E2	2-2	2	4-5	4.2	17-17	17	11-14	12.2	72-90	77.4
	E3	2-2	2	3-4	3.8	17-26	18.8	8-13	10.2	40-81	58.4
G2	E1	0-0	0	2-5	3.8	0-8	2.8	0-8	2.4	24-48	33.2
	E2	0-0	0	1-2	1.4	8-11	9	4-10	7.4	42-77	55
	E3	0-0	0	1-2	1.2	6-14	10.2	4-13	7.8	30-80	52.8
G3	E1	0-0	0	9-14	10	7-16	8.8	7-14	8.4	48-60	51.2
	E2	0-0	0	5-13	9	7-57	27.2	5-20	13.2	48-122	85.2
	E3	0-0	0	5-8	6.4	15-37	24.6	10-26	16.8	60-119	84
G4	E1	3-4	3.4	8-12	10.8	39-100	63.8	15-46	28.8	182-440	267.2
	E2	3-4	3.4	6-9	7	38-107	64.6	17-35	21.6	168-288	219.2
	E3	3-4	3.4	4-10	6.4	36-88	56.4	14-34	23.4	168-270	223.6
G5	E1	0-0	0	3-4	3.6	24-24	24	7-12	8.6	132-176	151.8
	E2	0-0	0	0-2	0.4	25-32	28.8	8-17	11.6	132-168	152.8
	E3	0-0	0	0-0	0	25-30	26.8	7-14	9.6	132-156	147.8
G6	E1	6-6	6	52-65	58.4	144-212	167.8	40-55	50.4	360-522	423
	E2	6-6	6	35-60	47.4	144-167	154.6	39-76	58	384-420	404
	E3	6-6	6	35-46	40.6	162-250	193.6	34-94	67.8	384-621	456

Table 3. Overall results of the experiments compared to the quality of the initial drawings.

Experiment	Crossings	Dummy Vertices	Bends	Area
	Av	Av	Av	Av
E1	80.3%	126%	103.4%	124.8%
E2	44.0%	195%	143.9%	151.0%
E3	35.0%	185%	157.1%	151.1%

Table 4. Usage of the main operations.

Exp.	Select	Move	Layering	Ordering	Return to Best Sol.	Add Constr.	Delete Constr.	Other
Total	10.6%	10.7%	9.7%	60.2%	2.2%	4.3%	1.3%	1%
E1	0%	0%	10.7%	76.3%	2.4%	7.7%	2.5%	0.4%
E2	16.2%	0%	11.3%	64.6%	1.9%	4.5%	1.1%	0.4%
E3	15.5%	33.3%	6.9%	39.1%	2.4%	0.6%	0.1%	2.1%

we call idle time, the system was stopped, waiting for the user to do some action. During that time the user was thinking about what kind of hint to give to the system. This indicates that there is much CPU power left for improving the cooperation between the system and the user in our approach. We envision a more collaborative framework where the system may work in background improving the results.

6 Conclusion

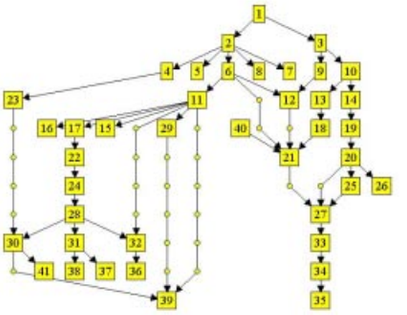
User hints, particularly focus and manual changes helped an optimization process based on the Sugiyama method to improve drawings of directed graphs. Some effort was demanded in order to adjust the traditional graph drawing method to allowing user interaction. However, the system with this facility seems more attractive and powerful than just a simple manual post-processing of the drawings. We are now improving our system based on the results obtained from the pilot study. The constraint mechanism is being revised to include new constraints that can represent the users desires better. We are also investigating the use of better optimization methods such as meta-heuristics and exact methods. For such methods, hints can provide a good way of reducing the space of solution. Moreover, we believe that focus and constraints can be implemented in a more straightforward way in those methods.

References

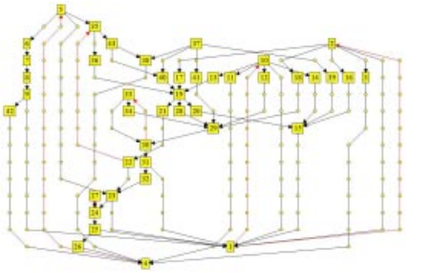
1. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Graph drawing: algorithms for the visualization of graphs. New Jersey: Prentice-Hall, 1999.
2. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. IEEE Trans. Syst. Man Cybern., SMC- 11(2):109-125, 1981.
3. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. IEEE Trans. Syst. Man Cybern., SMC- 18(1):61-79, 1988.
4. K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. IEEE Trans. Softw. Eng., 21(4):876–892, 1991.
5. Proceedings of the 22nd Annual Conference on Software Engineering (ICSE 2000). Limerick, Ireland, June 4-11, 2000.
6. Graph Drawing Contest. In Proceedings of the 8th International Symposium on Graph Drawing, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000. ISBN: 3-540-41554-8.
7. R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.
8. J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In Proc. of The International Conference on Imaging Science, Systems, and Technology (CISST'98), pp. 154-160, Las Vegas, Nevada: CSREA Press, July 6-9, 1998.

9. H. A. D. do Nascimento, P. Eades and C. F. Xavier de Mendonca Neto. A multiagent approach using A-Teams for Graph Drawing. In Proceedings of the 9th International Conference on Intelligent Systems, Louisville, Kentucky - USA, June 15-16, 2000, pag 39-42.
10. D. Andersen, M. Andersen, M. Lesh, J. Marks, B. Mirtich, D. Ratajczac and K. Ryall, Human guided simple search, to appear in the proceedings of the annual conference of the American Association for Artificial Intelligent, 2000.
11. N. Lesh, J. Marks, and M. Patrigname. Interactive Partitioning. Graph Drawing Conference, 2000.
12. K. Ryall, J. Marks, S. Shieber. An interactive constraint-based system for drawing graphs. In Proc. of the ACM Symposium on User interface Software and Technology (UIST' 97), pages 97-104, Oct. 1997, Banff, Alberta.
13. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. Software-Practice and Experience, vol. 21, no. 11, 1129-1164, 1991.
14. K. Dauner. "Ein interaktiver Genetischer Algorithmus fur das Zeichnen von Graphen".("Interactive genetic algorithm for graph drawing"). Diplomarbeit (Master Thesis), Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, 1997.
15. A. E. Jacobsen. "Interaktion und Lernverfahren beim Zeichnen von Graphen mit Hilfe evolutionarer Algorithmen" ("Interaction and learning methods for graph layouts with the help of evolutionary algorithms"). Diplomarbeit (Master Thesis), Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, 2001.
16. E. Koutsofios and S. North. Drawings graphs with dot. Technical Report, AT&T Bell Laboratoties, Murray Hill, NJ, USA, Sep 1991.
17. K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. Conference proceedings on Empowering people: Human factors in computing system: special issue of the SIGCHI Bulletin, pages 43 - 51,1990.
18. P. Eades. A Heuristics for Graph Drawing. Congr. Numer., 42, 149-160, 1984.

Appendix – Examples of Drawings Produced by the Users



(a) Unix System Family (G5)



(b) Forrester's World Dynamics Graph (G6)