A Framework for Human-Computer Interaction in Directed Graph Drawing

Hugo A. D. do Nascimento

Basser Depart. of Computer Science, University of Sydney, Australia
Email: hadn@cs.usyd.edu.au
Supported by UFG and CAPES-Brazil, and by The Australia Research Council.
(Ph.D. Scholarship from CAPES).

Abstract

This paper describes some studies in Human-Computer Interaction for Directed Graph Drawing. We have developed a system where users can help some standard graph drawing algorithms to produce nice drawings of a graph according to a set of aesthetic criteria. The system follows a general framework for interaction with optimisation processes that can be applied to many optimisation problems. Some discussion about the framework and possible improvements is presented.

Keywords: human-computer interaction, user hints, optimisation.

1 Introduction

Graph Drawing is an emerging research area with strong applications in information visualization. The main aim of Graph Drawing is to produce "nice" visualizations of graphs[1] that display the relationships between the entities. Even though the concept of "niceness" is inherently subjective, there are some commonly accepted measurable properties of "nice" drawings of graphs [Battista et al., 1999, Purchase, 1997]. These properties, called *aesthetic criteria*, include: *show few edge crossings*, *show monotone edge orientation*, and *display symmetry*. These criteria have been shown to improve the readability of diagrams [Purchase, 1997].

Unfortunately, satisfying aesthetic criteria involves some difficulties: producing the best drawing according to a single aesthetic criterion is very often an NP-hard problem, and several criteria may conflict as well. For instance, drawing general directed graphs with no edge crossings (i.e. a planar drawing) and showing uniform edge orientation is an NP-hard problem [Garg and Tamassia, 1995]. As a consequence, only some sort of compromising solution can be obtained.

The graph drawing activity is, in fact, a multi-objective optimisation problem, where two or more objectives conflict and cannot be solved optimally. For this kind of problem, heuristics must be used. The Graph Drawing literature contains many heuristic methods for drawing different types of graphs. In most of the cases, these heuristics produce good drawings in a reasonable amount of time. However, when the quality of the computed drawing is not good enough for our needs, few options are left in order to improve it; the most traditional approaches are to redraw the graph using another method, and to make the necessary adjustments manually. Some flexible improvement methods, such as Genetic Algorithms and Simulated Annealing, can be used, but experience has shown that they are in general very slow and hard to tune for the Graph Drawing problem [Davidson and Harel, 1996, Branke et al., 1991, Battista et al., 1999].

In this paper, we present an approach to improve drawings of graphs that differs from the tradition "do it manually" or "try another heuristic". We allow a continuous interaction between an automatic graph drawing method and its user, such that the drawing can be improved in a more natural way. At first, an initial drawing of the graph is automatically computed by the system. Then the user drives the optimisation, running the graph drawing algorithms only on parts of the drawing that need major improvement. Identifying areas of the drawing that are worse than others, as well as having some idea about how to improve them, seem to be easy for human beings. These tasks, however, are hard to implement as an algorithm. Therefore, we include the human element in the graph drawing optimisation process, aiming to take advantage of human skills. Some other interactive features complete our approach, such as manual changes of the drawing, and insertion of layout constraints. These forms of interaction are called *hints*.

This paper discusses how an interactive framework based on user hints can be applied to the problem of drawing directed graphs. It provides an overview of our previous work [do Nascimento and Eades, 2001], and adds some remarks on issues arising from experimenting with the framework.

The remainder of the paper is organized as follows: sections 2 and 3 describes our previous work; section 2 presents our general framework for interaction in optimisation processes; section 3 describes an interactive system for Graph Drawing and experimental results produced by a pilot study. Section 4 is new; it presents our remarks on the framework and points directions for improving the approach; section 5 describes some related work on interactive optimisation procedures; finally, in section 6 we conclude with an agenda for future development.

2 A Framework for Interactive Optimisation

In this section, we explain a framework where a user guides an optimisation method in order to improve a solution for a problem. User guidance occurs by giving "hints" to the optimisation method. *Hints* are pieces of information that help the method to reduce the search space, to escape from local minima, or to resolve ambiguity when there are more than one feasible solution. We consider three types of hints: focus, constraints, and manual changes of a solution. Next,

[1] A graph $G = (V, E)$ is a mathematical model consisting of a set V of vertices, and a set E of edges linking pairs of vertices of V. For a complete terminology of Graph Theory, see [Bondy and Murty, 1976]

we describe each type of hint in the context of 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 just some areas that do not satisfy the desired aesthetic criteria very well. By identifying these areas, the user can restrict the drawing algorithm to redraw only the focused areas.
- **Layout Constraints.** Layout constraints are useful 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 is required 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 performs manual changes only on vertices, by moving them. Changes on edges can be done by moving their related vertices. Manual changes are already commonly used in graph drawing activities as part of a post-processing and fine-tuning step. However, here we use them as a more powerful tool, since changes in a drawing may drive the system out of a local minimum to a better solution.

The framework based on hints is illustrated in Figure 1. The optimisation method is an improvement algorithm that works iteratively on a solution (a drawing of a graph). The system always keeps two solutions: the current **working solution** and the **best solution** computed up to now. The user gives hints to the method and calls it to improve the working solution. The method then creates a new solution that may or not be better than the current one. Nevertheless, every new drawing created is analysed and compared to the best solution. If the new solution is better than the current best solution, the best solution is immediately updated with the new drawing.

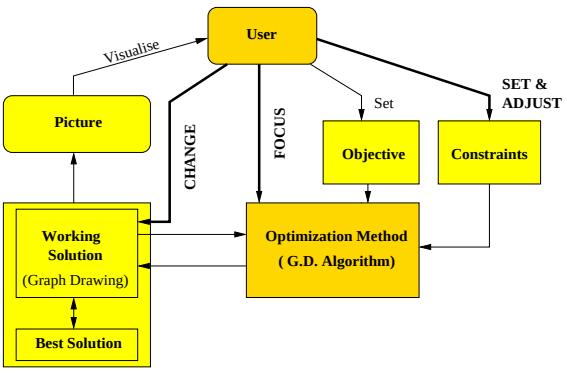


Figure 1: An interactive framework for optimisation processing based on user hints. Hints are represented by thick arrows with uppercase labels. Note that all types of hints are input to the optimisation method, even if indirectly.

The system provides continuous feedback of its state to the user, in the form of pictures, text and sounds. The feedback highlights quality aspects of the solutions. Based on the feedback, the user can decide what kind of hint to give next to the optimisation method.

3 GDHints: a Hints-Based Graph Drawing System

Here we describe an interactive system, called *GDHints*[2], we developed for drawing directed graphs. The system implements the framework presented in the previous section. It allows the user to define focus and layout constraints, as well as to perform manual changes of a drawing through a graphical interface. A snapshot of *GDHints* is shown in Figure 2.

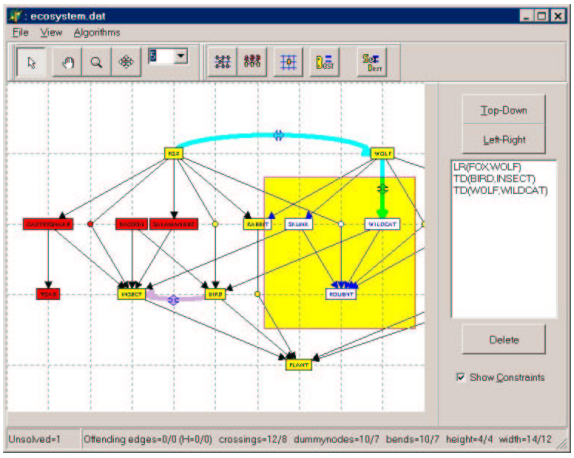


Figure 2: GDHints – an interactive system based on user hints. In the system, vertices are represented by boxes, and edges by thin arrows. Thick arrows describe layout constraints. Constraints are also indicated by a list of items on the right-hand side of the screen. For instance, in the drawing above, there are one *Left-Right* constraint and two *Top-Down* constraints. Selected (focused) vertices are presented in dark colour. The user can select a vertex by clicking on it. It is possible to select several simultaneously by specifying a rectangular region that involves them, as shown on the picture. The tool bar on the top of the screen contains the main graph drawing features. The status bar on the bottom shows the quality of the working solution compared to the quality of the best solution.

The optimisation method included in the system is a collection of heuristics based on the Sugiyama method for drawing directed graphs [Sugiyama et al., 1981]. 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 [Sugiyama et al., 1981, Battista et al., 1999]:

1. **Cycle Removal** - reverses some edges of the graph in order to make it acyclic. We adapted the Greedy-Cycle-Removal heuristic from [Battista et al., 1999] for this aim.
2. **Layer Assignment** - assigns vertices 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 short edges linking dummy vertices. We use a version of the Longest Path Layering heuristic [Eades and Sugiyama, 1990] for the layering assignment.

[2] *GDHints* can be downloaded from http://www.cs.usyd.edu.au/~visual/systems/gdhints

3. **Crossing Reduction** - sorts vertices in each layer in order to reduce the number of edge crossings. This is done by sweeping the layers upward and downward repeated times and apply the barycenter heuristic [Sugiyama et al., 1981] in each stage.

4. **Horizontal Coordinate Assignment** - sets the X-coordinate of each vertex such that edges are as straight as possible, and bends and the width of the drawing are minimized. All edges changed in step 1 are also reversed to their original orientation in this step.

We preserve the general structure of the Sugiyama method and adjust each step to support focus and layout constraints. Given a graph $G = (V, E)$, we *focus* on a selected set $A \subseteq V$, by applying the Sugiyama method only on A. The X, Y coordinates of all vertices in $V - A$ are kept fixed. On the other hand, layout constraints are modelled either as extra edges added to the graph or as normal constraints that impose an ordering to vertices. A detailed description of the changes of the Sugiyama steps for dealing with focus and constraints is presented in our previous paper [do Nascimento and Eades, 2001].

The system groups the Sugiyama's steps into two main procedures:

- **Layering** - executes the Cycle Removal and the Layer Assignment steps;
- **Ordering** - executes the Crossing Reduction step.

The Horizontal Coordinate Assignment step is not explicitly implemented in our system. It is partially carried out by some heuristics in the Crossing Reduction step that already assign a good X-coordinate to vertices. Moreover, the Horizontal Coordinate Assignment can be improved by iteratively calling the Ordering procedure.

3.1 User-System Interaction

The aim of the interaction between the user and the system is to improve a drawing of a graph according to the following priority of aesthetic criteria: (1) few upward and horizontal edges (that we call *opposite edges*), (2) few edge crossings, (3) few dummy nodes, (4) few edge bends and (5) small drawing area.

At the beginning, the layering and the ordering procedures are automatically executed for producing an initial drawing. Then the user starts by giving hints to the system and calling the layering and the ordering algorithms again.

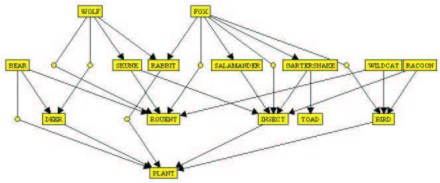
Part of the tasks of the system is to detect and automatically save the best drawing generated so far. At any time, the user can return to this drawing or can force the system to accept the current working drawing as the best one.

The system also provides useful feedback for user actions. This highlights some bad quality aspects of the drawings using colours. Moreover, every time a better solution is found, the system responds by playing a sound and animating an icon.

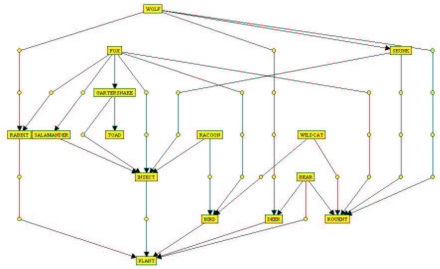
3.2 Experimental Results

We did a pilot study in order to verify whether users could help the system by giving hints. The study involved five users in three types of experiments. All users had a background in Computer Science and in Graph Drawing. The experiments types were:

- **E1**: the users could provide only constraints to the system. The ordering and layering procedures ran on the whole graph.



(a)



(b)

Figure 3: Drawings of the predator-pray ecologic system. Drawing (a) has 16 crossings, 7 dummy nodes, 7 bends, width 12 and height 4. Drawing (b) has 5 crossings, 33 dummy nodes, 17 bends, width 15 and height 7.

- **E2**: the users could provide constraints and focus.
- **E3**: the total functionality of the system was available. The users could give any kind of hints, including constraints, focus and manual changes.

Six graphs were used in the experiments. The users performed each type of experiment for each graph. Twenty minutes were allowed for each drawing activity. In the total, we had about 30 hours of experiments - 6 graphs x 3 experiments x 20 minutes x 5 users.

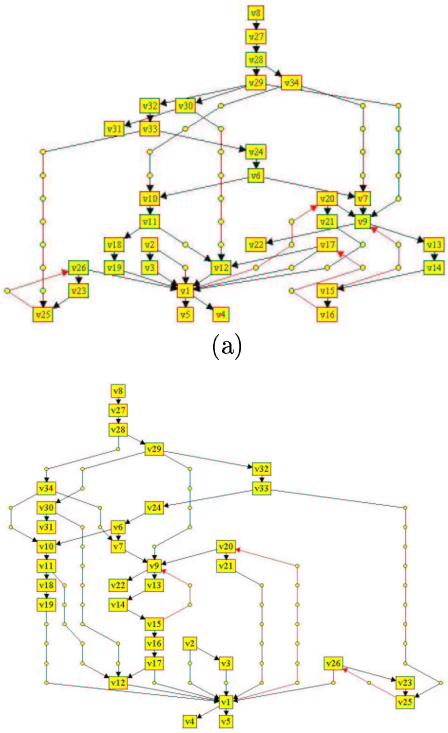
Figures 3, 4, 5, and 6 show drawings of the graphs. The drawings labelled *(a)* are the initial solutions created by the system. They are the same for all the five users and the three types of experiments. The drawings labelled *(b)* are example of improved solutions created by the users after interacting with the system for 20 minutes.

The experiment size is too limited to describe precise relations between all operations carried out by the users and improvements of the drawings. However, some general trends about how hints affected the aesthetic criteria could be perceived with the pilot study.

Basically the main improvement of the initial drawings was on the number of edge crossings. On average, the users were able to reduce 20% of the number of crossings in experiment E1, 55% in experiment E2 and 60% in experiment E3. The number of opposite edges, on the other hand, was not improved much. This is because the layering algorithm already produces a result very close to the optimum. The number of dummy vertices and bends, and the area of the drawings were higher than the initial values for almost all experiments. This shows that there is a trade-off between such aesthetics and edge crossings. Even though this trade-off was observed, the final drawings appear to be more readable than the initial layouts.

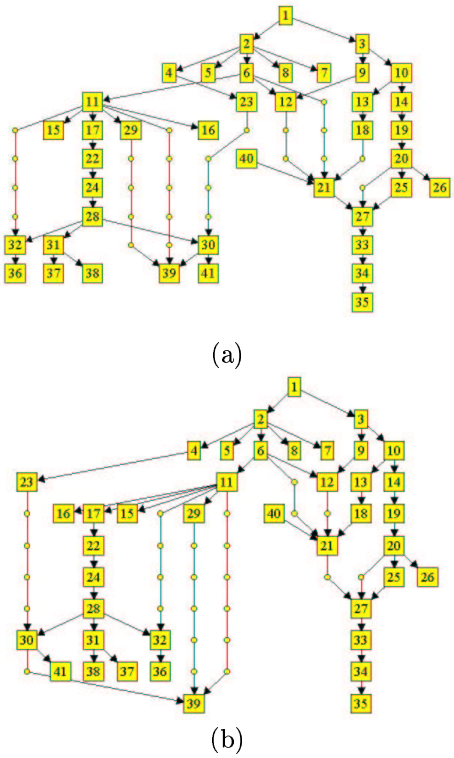
4 Remarks

In this section, we present our main remarks about the GDHints system and the interactive framework. We also suggest some ideas for improving the work.



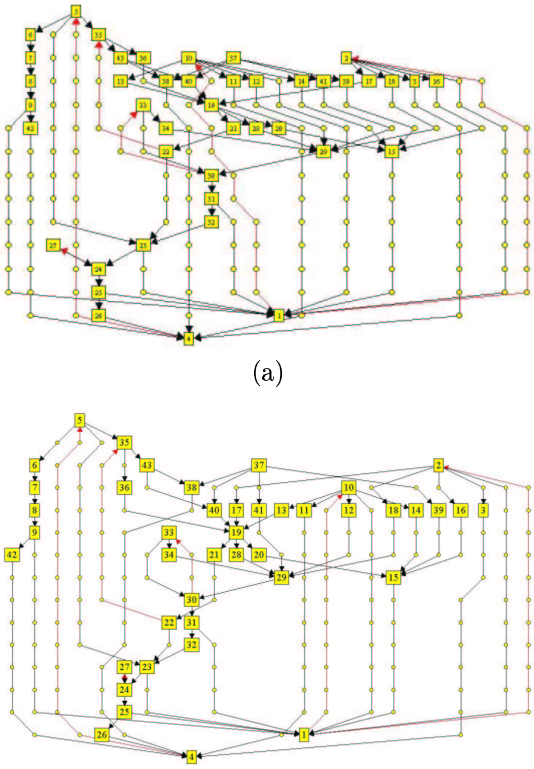
(a) (b)

Figure 4: Drawings of the C language syntax graph. Drawing (a) has 4 opposite edges, 12 crossings, 39 dummy nodes, 19 bends, width 13 and height 14. Drawing (b) has 3 opposite edges, 4 crossings, 61 dummy nodes, 31 bends, width 15 and height 18.



(a) (b)

Figure 5: Drawings of the Unix family-tree graph. Drawing (a) has 4 crossings, 24 dummy nodes, 9 bends, width 12 and height 11. Drawing (b) has no crossing, 25 dummy nodes, 7 bends, width 13 and height 11.



(a) (b)

Figure 6: Drawings of the Forrester's World Dynamics Diagram. Drawing (a) has 6 opposite edges, 70 crossings, 144 dummy nodes, 56 bends, width 24 and height 15. Drawing (b) has 6 opposite edges, 35 crossings, 162 dummy nodes, 55 bends, width 24 and height 16.

4.1 Constraints

In the system, constraints are used only as a tool for improving the aesthetic quality of graph drawings. This means that the system assigns a higher priority to the aesthetic criteria than to the constraint satisfaction, when comparing two solutions. The reason for this was to experiment with how constraints could help a traditional graph drawing problem, i.e.: avoiding upward and horizontal edges, minimizing the number of edge crossing, and other aesthetics.

In fact, the users found it hard to improve drawings using only constraints (in experiment E1). We observed that, by inserting a constraint and allowing the system to redraw the whole graph, the resulting new drawing can be very different from what is expected. For instance, a user inserts a *Left-Right* constraint, aiming to swap the position of two vertices that are close to each other in a layer. The drawing generated by the system may have the ordering of the specified vertices reversed, but the constraint also moves the vertices apart one from each other. As a consequence, the final drawing looks much worse than the initial one. We believe this problem could be solved by:

- providing more meaningful types of constraints that help the user to describe his or her intentions - eg. proximity constraints between vertices; and
- improving the ordering procedure for automatically refining the drawing (minimizing edge crossings) after solving constraints.

Constraints may also affect the position of vertices of the graph that are not directly restricted by any constraint, thus destroying the mental map of the

drawing. There are some mechanisms for preserving mental map explicitly. One solution is to assign a different degree of movement to distinct vertices; the higher the degree, the farther the vertex can be moved to. Based on this idea, vertices with many unsatisfied user-constraints would be set with the highest degree of movement. Moreover, extra constraints could be automatically created by the system for keeping the relative position of vertices that are not involved in any user constraint. Some work in this direction can be found in [Böhringer and Paulisch, 1990].

4.2 Focus

Experiment E2, based on focus and constraints, demonstrated a huge reduction of edge crossings when compared to the results from experiment E1. Note that the difference between experiments E2 and E3 is much smaller than between experiments E1 and E2. This indicates that focus plays a very important role in the crossing minimization processing. The main advantage of having focus is to concentrate the graph drawing algorithms on parts of the graph that really have to be redrawn. Moreover, by focusing on a small set of vertices, we can reduce the execution time of our algorithms.

Another issue is that, when focus is allowed in the system (in experiments E2 and E3), we can simulate a manual approach for preserving mental map. This is possible by selecting only vertices directly involved in constraints and redrawing them. After that, some refinement can be achieved by selecting vertices near the constrained ones and calling the ordering procedure again. This manual approach can be implemented later as an automatic process without much effort.

4.3 User Search Method

The system allows a search method based on local exploration with backtracking. The main idea is to improve a solution by exploring its neighbourhood (i.e. looking for a better solution that can be generated by applying a small sequence of changes to the current drawing). When a path of investigated solutions does not lead to any good improvement, the user can backtrack to the start point and explore a different path. This is possible with the features for returning to the best computed solution and setting the current working solution as the best one.

We perceived that the users intuitively applied this search method in their experiments. They initiated the search by using the best computed drawings as a starting point. Then they changed the drawing several times using hints. If no better drawing was found after a number of iteration, they returned to the best solution and started the search again. However, if a better solution was found during the exploration, then the new drawing automatically replaced the current best one and became the start point. Sometimes the users did not find a better solution with the search, but the working drawing seemed to be promising. So they forced the system to set it as the start point (the best solution). This allowed concentrating on the promising drawing, rather than on the previous best drawing.

4.4 Quality Feedback

The icon animation and the sound event seemed to have effected the user in two different ways:

- they indicated that the action made by user was correct and, consequently, reinforced a future usage of the same operation;
- they motivated the user to keep working on the graph drawing problem.

We believe that such kind of animations and sound may help optimisation tasks. There is much space for developing better feedback strategies. One possible idea is to build intelligent agents that help to identify promising areas of the problem and/or indicates whether the user is on the right path towards a good solution.

4.5 Task Division

The advantage of using an interactive framework for optimisation problems is the possibility of getting better results via some sort of division of tasks between the human and an automatic search method. In our case the system automatically identifies and updates the best drawing according to a hill-climbing approach. This frees the user of having to analyse the quality of every new drawing, so that him or her can focus on the improvement task. As we will see in section 5, this is similiar of the task division proposed in [Anderson et al., 2000].

Another interesting point is that, on average, only 10% of the experiment time was spent by the system doing some useful activity. In the other 90% the system was idle, waiting for user hints. This also indicates that there is space for improving the system. We believe that it may be possible to take advantage of better optimisation methods that run in background, doing some processing in parallel with the user.

4.6 Fully Automatic versus Interactive Systems

One may ask why invest efforts for an interactive framework. Why not to develop better algorithms that do what the user does? We may think that, if the human being is necessary during the optimisation task, then the system is not good enough and, therefore, it should be improved. There are several answers for this question and we do not intend to cover them extensively here. We believe that the two approaches - fully automatic optimisation and interactive optimisation - are not mutually exclusive. Evidently, there are applications where it is necessary to provide a solution without user interaction. Example of such applications are CASE and CAD tools that use Graph Drawing methods as a secondary task and where the flexibility of having different drawings is small or not important. In those cases, the system has to be able to produce a drawing of a graph by itself.

In some other cases, interaction may contribute. This happens when:

- the graph drawing activity itself becomes the main goal (such as in VLSI design or Graph Drawing tools);
- the problem is dynamic;
- the application is domain dependent and the domain is not completely known prior running the system; and
- there may not be enough resources (hardware, software or even financial resources) for developing and implementing a very efficient fully-automated software.

For those reasons user interaction can add more flexibility and capability to existing systems.

Nevertheless, allowing interaction does not prohibit us from improving automatic methods. Actually, by including better algorithms into our system,

we contribute to have better initial solutions. As a consequence, part of the job that used to be done by humans will now be carried out by the system. Note that, by having very good algorithms, it may even be possible to satisfy most of the users with just the initial drawing. Nevertheless, if there is still a need for improving the drawing, then such improvement is expected to be more difficult than before (when only simple algorithms were available). Again, user interaction will benefit from having better algorithms, since they can be re-executed on the remaining optimisation problems by using focus.

4.7 Learning from Observation

In an ultimate application, our interactive framework can be used as an environment for studying how users solve graph drawing problems. Through out observation it is possible to identify solutions for graph drawing problems that may lead to the development of new automatic algorithms.

4.8 A New Type of Hint

Finally, we have been considering to divide a problem into sub-problems as new type of hint that can be provided by the user. Such hint can be useful for defining a fine-grain degree of dependency between variables involved in the problem. This was done in a general way by using focus, when the users divided the vertices of the graph into two groups: fixed and focused vertices. The result was a reduction of the problem complexity. We aim now a more refined subdivision, probably using clustering for highlighting the relationship between groups of vertices.

5 Related Work

Our approach is similar to the cooperative paradigm "Human-Guided Simple Search" (HuGSS), presented by Anderson et al. in [Anderson et al., 2000]. HuGSS divides an optimisation 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 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. Focus consists of setting search priorities for parts of the solution and/or defining how deep the search will be executed. Three levels of priority can be assigned to the elements of the solution:

- **low**: elements marked as low priority are not changed by the system;
- **medium**: elements marked as medium priority may be changed if necessary, but in a small degree of freedom;
- **high**: elements marked as high priority can be changed completely by the system.

Note that this priority scheme is more refined than our binary selected/fixed focus approach. We believe that even more than three levels of priority can be used, as proposed by the new type of hint described in the previous section.

The HuGSS paradigm was successfully applied to the problem of capacitated vehicle routing with time windows. It showed that a simple hill-climb algorithm could be significantly enhanced by user interaction. The same idea was applied later for graph clustering in [Lesh et al., 2000].

Focus, constraints, manual changes and clustering are also present in an interactive single-line diagram editor developed by Paris [Paris, 1998]. The editor serves as a user interface to a real-time power system simulator. It combines user actions with automatic graph drawing tools based on an adaptation of the Sugiyama method. The aim of the editor is to improve the quality of drawings or adjust them according to domain knowledge not present in the system. The user can perform manual changes and run automatic tools on a selected part of the drawing. Undo and constraint insertion facilities are available.

Another application of user hints is described in [Bayazit et al., 1999], where users help an automatic motion planner to solve motion planning queries. An interactive environment using haptic and visual interfaces is used for creating user-generated paths. Then these paths are automatically transformed into collision-free paths by the system. The idea behind the approach is that human beings can discover "critical" situations that are hard for the system to identify, and can provide approximate solution for them. Experiments with the interactive motion planner showed that it was able to produce better results and in less processing time than a fully automatic system.

These pieces of research show a trend for having users guiding optimisation processes. Our approach is another example in this direction.

6 Conclusion

Our experiments with an interactive framework showed that drawings of graphs could be significantly improved by having users giving hints to traditional graph drawing methods. Even though the heuristics we used are deterministic and very limited in exploring the space of solutions, their functionality was extended by the focus mechanism. Constraints and manual changes helped to convey the system to some specify solutions. Constraints, in particular, were hard to use when applied alone.

We have many plans for future extension of our approach. They are mainly experiments on how the framework would work for other optimisation problems with more complex search methods. Some specific studies we intend to do next are:

- try hints with meta-heuristics and exact methods;
- develop more meaningful sets of constraints as well as better ways of satisfying them; and
- explore new types of hints, which may be necessary for different optimisation problems.

References

[Anderson et al., 2000] D. Anderson, E. Anderson, N. Lesh, J. Marks, B. Mirtich, D. Ratajczac and K. Ryall. (2000). Human guided simple search. To appear in the *Proceedings of the annual conference of the American Association for Artificial Intelligent.*

[Battista et al., 1999] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. (1999). *Graph drawing: algorithms for the visualization of graphs.* New Jersey: Prentice-Hall.

[Bayazit et al., 1999] O. B. Bayazit, G. Song, and N. M. Amato. (1999). Providing Haptic 'Hints' to Automatic Motion Planners. In *Proc. of the Phantom User's Group Workshop(PUG99).*

[Bondy and Murty, 1976] J. A. Bondy and U. S. R. Murty. (1976). *Graph Theory with applications.* Macmillan: London.

[Böhringer and Paulisch, 1990] K.-F. Böhringer and F. N. Paulisch. (1990). 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 42-51.

[Branke et al., 1991] J. Branke, F. Bucher, and H. Schmeck. (1997). Using genetic algorithms for drawing undirected graphs. In J. T. Alander, editor, *Proceeding of the Third Nordic Workshop on Genetic Algorithms and their Applications (3NWGA)*, 193-205.

[do Nascimento and Eades, 2001] H. A. D. do Nascimento and P. Eades. (2001). User hints for Directed Graph Drawing. To appear in the *Proc. of the Graph Drawing Conference 2001.* Vienna, Austria.

[Davidson and Harel, 1996] R. Davidson and D. Harel. (1996). Drawing graphics nicely using simulated annealing. *ACM Trans. Graph.*, 15, No. 4, 301-331.

[Eades and Sugiyama, 1990] P. Eades and Kozo Sugiyama. (1990). How to Draw a Directed Graph. *Journal of Information Processing*, Vol. 13, No. 4.

[Garg and Tamassia, 1995] A. Garg and R. Tamassia. (1995). On the computational complexity of upward and rectilinear testing. In R. Tamassia and I. G. Tollis, editors, Graph Drawing (Proc. GD'94), vol. 894 of *Lecture Notes Comput. Sci.*, pp. 286-297. Springer-Verlag.

[Lesh et al., 2000] N. Lesh, J. Marks, and M. Patrigname. (2000). Interactive partitioning. in *Proc. of the Graph Drawing Conference 2000.*

[Paris, 1998] G. Paris. (1998). Cooperation between interactive actions and automatic drawing in a schematic Editor. S. Whitesides (Ed.): GD'98, *LNCS 1547*, pp.394-402.

[Purchase, 1997] H. Purchase. (1997). Which aesthetic has the greatest effect on human. *Proc. 5th Int. Symp. Graph Drawing*, LNCS 1353, pages 248-261.

[Sugiyama et al., 1981] K. Sugiyama, S. Tagawa, and M. Toda. (1981). Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109-125.