User Hints for Optimization Processes

by

Hugo Alexandre Dantas do Nascimento

Master in Comp. Science. 1997

A thesis submitted to

The School of Information Technologies

The University of Sydney

for the degree of

DOCTOR OF PHILOSOPHY

November, 2003

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Hugo Alexandre Dantas do Nascimento

Sydney

10th November 2003

To my parents, for their love and motivation,

and to my daughter.

Acknowledgements

This thesis would not have been completed without the help of many people. I would like to express here my immense gratitude to all of them.

Prof. Peter Eades, my supervisor, persuaded me to investigate the field of Human-Computer Interaction, and provided a dynamic and enthusiastic environment for my research. Peter's talented way of solving problems also had a great effect on my view of the world; he set an example of dedication and commitment to scientific research that I want to follow in life.

My parents, my sisters and my girlfriend were with me all the way, despite the geographical distance. By phone and mail they cared for my health and studies, and motivated me to look after myself. One of my sisters, Hani, also came to Australia and spent almost a year with me. Their continuous support was important to the conclusion of this work.

During these four years of research I received a Ph.D. scholarship from CAPES-Brazil. I also had a salary from the Universidade Federal de Goiás – Brazil, where I hold a lecturer position. Without this financial support, covering my living expenses and the tuition fees of the university in Australia, my stay here would not have been possible. This is the kind of investment that really brings positive results to a nation, and I am pleased to see that these two institutions did it well. Furthermore, I want to thank the Australian Research Council for providing me with an annual research allowance and for supporting my participation in conferences.

Several researchers made valuable comments about the topics that I investigated. I would like to thank Dr. Seokhee Hong and Dr. Masahiro Takatsuka (from the School of Information Technology at The University of Sydney), and the international researchers with whom I had interesting discussions. I am also in debt to all the cartographers that participated in the evaluation of our prototype for the Map Labeling problem. These include Murray Godfrey, David Godfrey and Yan Tollis (from Sydway/Ausway), Graham Russel and his colleges (from UBD), and the team of experts under the supervision of Brian McLauchlan (from DIGO-Australia).

My thanks go as well to the administrative and technical staff of the University of Sydney, and of the University of Newcastle (where I spent my first year of studies before Peter Eades moved to Sydney). At the University of Newcastle, the administrative support of Diana Edwards, Debbie Hatherell, Aaron Scott and David Montgomery provided all help that I needed. I received similar support at The University of Sydney from Sharon Chambers and Josephine Spongberg, and from the "workshop guys" – Remo Di Giovanni, Arthur Scott, Witold Janus, Robert Calabrese and Allan Creighton. I also want to thank Eu Wee Tan, who worked as a research assistant and helped me to organize the human experiments with a genetic algorithm.

Very good friends shared their time with me in Australia. When I needed a break from my studies I used to socialize with them. This always refilled my motivation to work. I cannot name all these friends here, but certainly a few ones have to be mentioned: Tony and Dirce Mendonça, Des and Dawn Mcmeekin, Hui-Ju Cheng (Jessica), Bernd Scheumman, Anja Kemena, Uta Zander, Kim Wellard, Luzenira Brasileiro and Clerilei Bier.

Finally, I am grateful to Stephen Anthony, Tim Dwyer, Chris Ottrey and again to my supervisor and to Dr. Seokhee Hong for having reviewed this thesis.

Contents

List of Figures

List of Tables

Abstract

Innovative improvements in the area of Human-Computer Interaction and User Interfaces have enabled intuitive and effective applications for a variety of problems. On the other hand, there has also been the realization that several real-world optimization problems still cannot be totally automated. Very often, user interaction is necessary for refining the optimization problem, managing the computational resources available, or validating or adjusting a computer-generated solution.

This thesis investigates how humans can help optimization methods to solve such difficult problems. It presents an interactive framework where users play a dynamic and important role by providing *hints*. Hints are actions that help to insert domain knowledge, to escape from local minima, to reduce the space of solutions to be explored, or to avoid ambiguity when there is more than one optimal solution. Examples of user hints are adjustments of constraints and of an objective function, focusing automatic methods on a subproblem of higher importance, and manual changes of an existing solution. User hints are given in an intuitive way through a graphical interface. Visualization tools are also included in order to inform about the state of the optimization process.

We apply the User Hints framework to three combinatorial optimization problems: Graph Clustering, Graph Drawing and Map Labeling. Prototype systems are presented and evaluated for each problem. The results of the study indicate that optimization processes can benefit from human interaction.

The main goal of this thesis is to list cases where human interaction is helpful, and provide an architecture for supporting interactive optimization. Our contributions include the general User Hints framework and particular implementations of it for each optimization problem. We also present a general process, with guidelines, for applying our framework to other optimization problems.

CHAPTER 1

Introduction

1.1 Motivation

With the technological revolution that marked the last century, there was a general feeling that technology would soon be able to automate almost all kind of activities performed by humans. The concept of a *maid robot* (such as Rosie, the maid robot in the cartoon show “The Jetsons”) is a representative example of people’s expectations at that time. In fact, during the last fifty years of the twentieth century, technology did replace humans in many activities, such as the jobs in the automobile industry [128].

However, a new century has begun, and many automatic tools expected to exist by this time (including Rosie) remain as intangible goals[1]. In contrast to the technological predictions in the last century, we have now realized that several real-world problems are much more difficult than they seemed. Some of these difficulties appear in Artificial Intelligence, and in Image and Natural Language Processing; they are often associated with pattern recognition problems involving image, voice and video content. Another group of difficult problems comes under the heading of Combinatorial Optimization[2]; this is the focus of this thesis.

A broad variety of techniques have been developed for Combinatorial Optimization problems. They include heuristic strategies dependent on the problem, Dynamic Programming [16], Integer Programming [84, 140, 168], and meta-heuristic methods – such as Greedy heuristics, Simulated Annealing [2, 107], Tabu Search [78, 79], Genetic Algorithms [81], GRASP [65, 66] and Asynchronous Teams [41, 42, 159, 181]. Although a considerable amount of research has been done

[1]A small step towards a hard-worker and low-waged housekeeper has been made by researchers from MIT. It consists of a robot for cleaning floors called Roomba, which is available commercially since 2002. Information about Roomba can be found at *http://www.roombavac.com/* and *http://www.time.com/time/roomba/*.

[2]An optimization problem consists of finding the maximum or the minimum of a function defined on some domain. Furthermore, the solution of the problem usually has to satisfy a set constraints. The problem is said to be *combinatorial* if the domain if finite. A more detailed definition of a combinatorial optimization problem is given in Section 2.1.2.

in this area, many combinatorial optimization problems that have practical applications in the real world cannot yet be tackled satisfactorily via a fully automatic approach. Examples are bin-packing problems for the garment industry [85], steel, wood and glass cutting processes [15], vehicle routing problems [8], and timetabling [18, 83, 139, 197, 199]. The immediate reason for this lies in the computational complexity of these optimization problems. In the early 70's, Cook [35], Karp [103] and Levin [125] formulated an important theory that defines some problems as belonging to classes called *NP-hard* and *NP-complete*. This means that such problems are very difficult, and it is unlikely that they can be solved optimally in polynomial time[3]. As well as computational complexity, there are many other factors in the real world that contribute to the difficulty of a problem:

- The problem can be dependent on subjective domain knowledge. The domain knowledge may be difficult to express formally and may vary from person to person.
- Some characteristics of the problem may be unknown; in this case, the problem is potentially dynamic, since its objectives and constraints may need adjustment as the optimization progresses.
- The problem may include multiple objectives and constraints; optimizing only one of these aspects can be already an NP-hard problem; the challenge is then to consider all objectives and constraints simultaneously, and to find compromise solutions in conflicting situations.
- In general the problems studied in science are simplified models of real-world problems. As a consequence, most optimization methods available in the scientific literature do not handle all objectives and constraints that appear in practical cases.
- Finally, existing hardware technology (CPU power and memory size) may not be sufficient to deal with complex problems that contain several objectives or constraints, or have a large number of variables. Even when powerful computation resources do exist, they may be too expensive to acquire.

Complex problems, presenting one or more of the aspects above, are quite common in our daily life. Consider for example the job performed by a travel agent when an employee of a major company has to meet several customers in different cities in order to provide support or training for its products. The employee has to decide with the travel agent a flight itinerary that starts from his

[3]For more details about the theory of Computational Complexity, see Garey and Johnson [74].

or her current location, passes by each of the costumers' cities, and finally returns to the starting point. The company usually specifies some basic criteria for guiding this decision process: the usual rules are that the trip has to be as cheap as possible, and that the employee should not stay away for a long period of time. If the time allocated for serving each customer is fixed, and if they can be visited in any order, then the problem consists of minimizing the duration and the cost of the flights. This involves a number of different flight options in the case that more than one airline provides routes between the customers' locations; moreover, the airlines may have flights on different days of the week and with distinct time schedules. Figure 1.1 shows an example of a hypothetical network describing flights between six cities (labeled from A to F); each flight is represented by a line connecting two cities. The lines are labeled with the duration (in hours) and the cost of the flights. The duration represents the total flight time, from boarding to arrival and including intermediate stops in some cities not included in the diagram. Choosing the best itinerary for such a trip is in fact solving the *Traveling Salesman Problem* (TSP) [36] with two objectives. The objectives are to minimize the sum of the costs and the sum of the durations of the chosen flights. The TSP is a well known problem, and is NP-hard even for a single objective. Satisfying two objectives simultaneously is a more difficult problem.

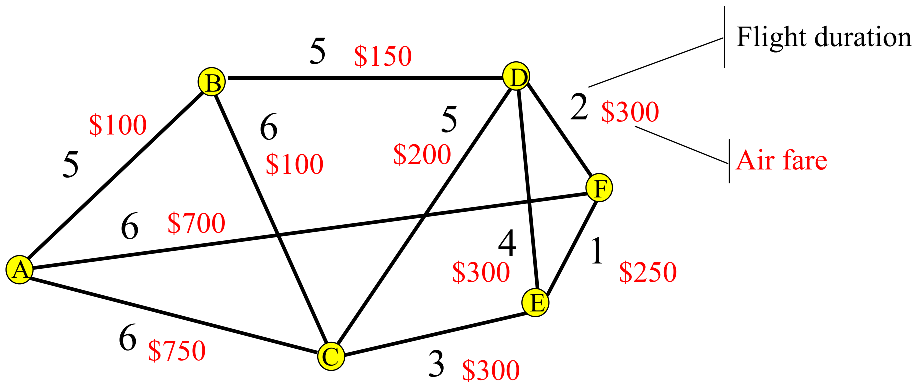


Figure 1.1: Duration and costs of flights between cities.

We can see that the problem to be solved by the employee and the agent is computationally difficult. Nevertheless, its entire complexity has not yet been discussed. Rather, we have only mentioned the company's interests and the basic knowledge about the costs and durations of flights. Many other elements are commonly involved in flight itineraries. For instance, some airlines may offer special discounts if a return ticket or a package of two consecutive flights is bought. In addition, the employee may have special requests such as: to give preference to flights that include meals in the basic package, or to fly with an airline that has a specific frequent flyer program.

Conflicts can also exist between the objectives; for example, shorter flights may imply higher costs. Consequently several compromise solutions exist, and the employee may decide subjectively on which option to take. As an example of a subjective decision, some employees may prefer to save the company's money by taking cheaper flights, which start early in the morning and have many connection stops.

Note that, if there is an emergency situation during the trip such that the employee has to prolong the stay in a particular city, then the itinerary should be amended without canceling all future flights.

Because of the enormous number of conditions and possibilities, many real-world combinatorial optimization problems still depend heavily on humans (in the "travel agency" problem above, the agent does most of the work). Computers can be used to search for possible initial solutions, but it is the human element that evaluates possibilities, recommends changes, and adjusts and approves the final solution.

In recent years, there has been an increasing interest in interactive tools for optimization methods. This is partially due to developments in User Interfaces, more generally in the area of Human-Computer Interaction, which now can provide intuitive and effective environments for interaction with a variety of applications. Human interaction in this context is beneficial since it provides a way of refining or adjusting the optimization problem to match the user's desires, or of managing the computational resources available. There are also well-known differences between human and machine skills for problem-solving, which can be exploited by human interaction. Computers, for example, are suitable for intensive computation, where many solutions can be created and numerically evaluated. Humans, on the other hand, are skilful in identifying patterns that differentiate good from bad solutions.

The need for having humans involved in optimization tasks was discussed by Donald Edward Knuth during his lecture in the Graph Drawing Conference in 1996[4], in Berkeley [145]. Knuth said:

"I also like to be able to tune things up later. . . I would urge all of you who are outputting the results of your graph drawing, not to just output a postscript file, but ideally you could output a file in a higher level language (and Metapost is the best I know), so that your users will be able to take that file and make slight refinements if they like afterwards, and rather easily."

Later, when talking about drawings of tree structures, which are frequently used in his famous

[4]A video of Knuth's talk, "Graph Drawing from a User's Perspective", can be obtained from the Mathematical Science Research Institute, at Berkeley-CA, USA. See *http://www.msri.org/index.html* for more information.

series of books *The Art of Computer Programming* [111, 113, 112], Knuth emphasized that:

"One system for drawing trees is not going to solve all the problems. Each one seems to have its own little thing that... If you didn't care about quality then you could get by with something that is totally automatic. But my idea is always to try to make something ninety-nine percent automatic, and then you can have fun with the other one percent. You get this self-satisfaction if you've added something and you haven't wasted a great deal of time on the extra thing. So I like people who design systems as if they are going to be fully automatic, but then they should also leave hooks so that people can toil with them easily afterwards; because visual aesthetics are something that I don't think we are ever going to totally quantify."

Knuth's comments on Graph Drawing show that it is exactly the type of complex optimization problem for which humans are still necessary. Therefore, he proposes that computational systems should be extended to support human interaction, allowing manual improvement of solutions when the algorithms cannot do it by themselves.

1.2 Aims

In this thesis we investigate the issues raised by Knuth, and show human interaction can contribute to optimization processes. More precisely, we investigate the following questions:

- In what circumstances is human intervention necessary?
- Can human interaction be done during runtime in order to improve the optimization?
- What is the best architecture for achieving this goal?
- Are fully-automatic optimization methods still useful in an interactive environment? If so, what methods are more suitable and how can they be adapted to support interactive facilities?

For answering these questions we present an architecture that we call the *User Hints framework*. In our framework, users can control an optimization process by providing *hints*, which are changes to the objectives and constraints of the problem, and direct control of the optimization process. Such changes allow the users to include domain knowledge, escape from local minima, eliminate ambiguous situations or speed up the optimization process.

It is important to note that many optimization systems that support human interaction implement this feature as a post-processing step in a sequential approach. Figure 1.2 shows the traditional framework with post-processing – which is also what Knuth has suggested: firstly an automatic method is applied, and then the human improves the computer-generated solution via a manual adjustment.

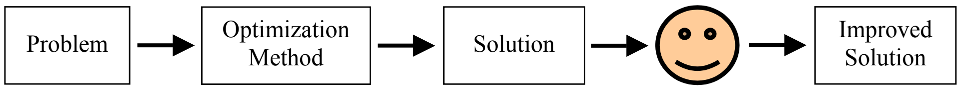


Figure 1.2: Post-processing improvement of a solution for an optimization problem.

The User Hints framework is different from the post-processing model, since it considers a stronger relation between the user, the automatic tool and the solution been improved. Figure 1.3 presents a general description of the User Hints framework. The automatic method acts as an improvement algorithm; not only can it be executed on the initial stage of the optimization (in order to produce a good initial solution), but can also be re-applied to improve solutions modified by the user. The users interact with the solution, the optimization method, and with the description of the problem. (Note that in the traditional post-processing framework the user can still return to the initial stage and change the problem or replace the optimization method; however, this implies a new optimization processing that does not take into consideration improvements done by the user on the previous solution.)

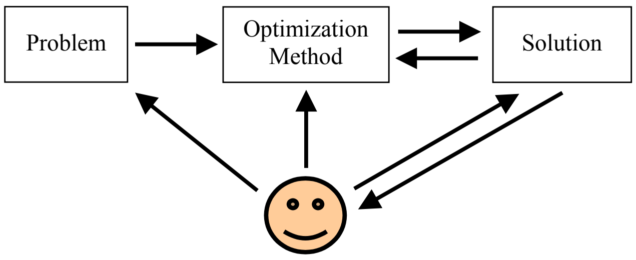


Figure 1.3: A general diagram of the User Hints framework.

We investigate the potential of the User Hints framework by applying it to some combinatorial optimization problems. We did not consider the "travel agency" problem. Instead, we approached three other more familiar problems: Graph Clustering, Graph Drawing and Map Labeling.

1.3 Research Methodology

The research methodology used during this work consisted of the following steps:

1. Investigating optimization problems that can benefit from human interaction.

2. Defining a general interactive framework for optimization.

3. Building prototype systems to test interaction facilities. This involves:

   (a) building specific interactive frameworks;

   (b) building systems in the application domain.

4. Evaluating these systems using:

   (a) domain experts,

   (b) controlled experiments, and/or

   (c) quality parameters.

5. Refining the general interactive framework based on the experience with the systems, and developing guidelines for applying it to other problems.

1.4 Contributions

The objective of this thesis is to investigate whether human interaction can help to produce high quality solutions in optimization processes. We present a framework for this goal, and describe derivations of it for three case studies. The case studies show how user hints, automatic methods and visualization tools can be put together for Graph Clustering, Directed Graph Drawing and for Map Labeling. The main advantages and disadvantages of the User Hints framework arise from the experiments done with our prototype systems.

The specific contributions of the thesis are listed below:

- A general architecture for human interaction in optimization processes.

- An approach for including user hints in Graph Clustering. This was our first study of user hints and we applied the lessons learned from our experience with Graph Clustering in other case studies.

- An approach for incorporating user-defined layout constraints and *focus* with the Sugiyama method, for drawing directed graphs. Focus consists of restraining the scope of the method to work only on a selected part of the optimization problem.

- A new genetic algorithm for drawing directed graphs; this algorithm supports layout constraints and focus; it implements a design for the solution representation and for the evolutionary operators that yields good drawing solutions.

- For the Map Labeling case, an approach for focus that constructs a new labeling problem based on selected elements (features and labels) of the map.

- A recursive operation that computes the maximum set of elements of a cartographic map that may need to be relabeled in order to improve the label position of a particular feature.

- A process and intuitive guidelines for applying the User Hints framework to other optimization problems.

The results in this thesis can benefit researchers and system developers in the case study areas, as well as in major fields such as Human-Computer Interaction and Combinatorial Optimization. The discussions made throughout the thesis together with the guidelines given in the General Remarks Chapter provide sufficient material for implementing User Hints frameworks for other optimization problems. For the Graph Drawing and the Map Labeling problems, the thesis already includes a good interactive approach that can be used as a starting point for future extensions.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 provides a background on previous interactive approaches for optimization. Chapter 3 introduces the User Hints framework. Chapter 4 discusses the application of user hints to the problem of clustering graphs. Chapter 5 investigates how our framework can be used to improve drawings of directed graphs with the Sugiyama Method. Chapter 6 extends the graph drawing investigation by presenting a genetic algorithm that supports layout constraints and focus. Chapter 7 applies the User Hints framework to the problem of labeling point-features in maps. Chapter 8 discusses additional issues related to user interaction, presents a process and guidelines for applying our approach to optimization problems, and discusses extensions to the framework. Chapter 9 draws our general conclusions and proposes suggestions for future research.

The thesis also has two appendices: Appendix A presents two interactive tables that we built for experimenting with human interaction – the tables are based on the *optimization table* developed at

MERL, Boston [8]; Appendix B describes the content of the CDROM included with the thesis. It also explains the user interface of our prototype systems.

CHAPTER 2

Background

This thesis concerns user interaction with optimization processes. The thesis uses well established concepts in Graph Theory, Combinatorial Optimization, Human-Computer Interaction and Information Visualization; brief introductions to these areas are given in Section 2.1.

The remainder of this chapter describes examples of systems and approaches that involve collaboration between humans and computers working together to solve problems. Of course many thousands of systems involving human-computer collaboration have been designed since the advent of computers. We have restricted our attention to those which have some implications for the thesis.

In Section 2.2 we discuss human interaction in three general categories: Computer Aided Graphics Design, Information Retrieval and Mixed Initiatives.

In Section 2.3, we are more specific: we consider approaches and systems in which humans collaborate with optimization algorithms.

Section 2.4 brings the background together by summarizing the roles usually performed by humans and computers in interactive systems, and presenting two main goals for having human interaction in optimization processes. This leads to the development of the User Hints framework.

2.1 Basic Concepts

2.1.1 Graphs

A graph is a mathematical model widely used to describe relationships between entities. The definitions below follow the terminology used in [23, 179].

A (*undirected*) *graph*[1] $G = (V, E)$ consists of a finite set of *vertices* V and a finite set of *edges* E. An edge $e \in E$ is an non-ordered pair (u, v) of vertices of V. We say that u and v in V are

[1]Also called *general graph*.

adjacent or *neighbors* if there is an edge $e = (u, v) \in E$. In that case, we also say that e is *incident* to u and v, and that u and v are the *endpoints* of e. The *degree* of a vertex $u \in V$ is the number of edges in E incident to u. A *loop* is an edge with one endpoint.

A *path* from a vertex u to a vertex t in $G = (V, E)$ is a sequence $(u = v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_{k-1}, v_k), v_k = t)$ where (v_i, v_{i+1}) are edges of E (for $i = 0, 1, \ldots, k-1$) and $v_0, \ldots, v_k$ are vertices of V. Without loss of generality, we can omit the edges in the sequence. When a path has both extreme vertices the same ($v_0 = v_k$), it is called *a cycle*. The *length* of a path is the number of edges in the sequence. The (graph theoretic) *distance* between two vertices u and v in the graph is given by the shortest path from u to v.

A graph G is *connected* if there is a path between all pairs of distinct vertices u and v in G.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The subgraph G' is an *induced subgraph* of G if, for all $e = (u, v) \in E$ with $u, v \in V'$, $e \in E'$.

A *tree* is a connected graph that has no cycles.

Similarly, a *directed graph* $G = (V, E)$, consists of a finite set V of vertices and a set E of directed edges with *ordered* pairs of vertices of V. An edge $e = (u, v) \in E$ is said to be an *outgoing edge* from u and an *incoming edge* to v. Note that the the terms (u, v) and (v, u) define two different edges in a directed graph, while they represent the same edge in an undirected graph. For a vertex $w \in V$, we represent the number of incoming edges to w by *indeg*(w), and the number of outgoing edges from w by *outdeg*(w). The (*total*) *degree* of w is *indeg*(w)+*outdeg*(w). A vertex is a *source* if it has no incoming edge; it is a *sink* if it has no outgoing edge.

The concepts of path and graph connectivity are similar for both undirected and directed graphs.

2.1.2 Combinatorial Optimization

We use notations from [71, 86, 138] in this section.

In general, an *optimization problem* can be described as maximizing an *objective function* $f : D \to \Re$ subjected to a set C of constraints on D, where D is the *domain* or *search space* of the problem. This definition describes a *maximization problem*; a *minimization problem* is similar. An optimization problem is *multi-objective* if it has two or more objective functions to be maximized/minimized.

The optimization problem is *combinatorial* if the domain D is finite. The elements of D are in general multidimensional vectors that represent variables of the problem. We often refer to the elements of D informally as *solutions* of the problem.

Let $S \in D$ be the set of all vectors x that satisfy the constraint set C. The problem is said to be *feasible* if $S \neq \emptyset$; otherwise, the problem is *infeasible*. All elements of S are *feasible solutions* to the problem. An element $x \in D$ is *infeasible* if $x \notin S$. The *global optimum* or *optimal solution* is a solution $x^* \in S$ such that $f(x^*) \geq f(y)$ for all $y \in S$.

Many optimization methods proceed by changing an initial solution to another solution by a variety of operations. A collection of operations on D is *elementary* if it has the following properties:

(a) If $y \in D$ can be obtained from $x \in D$ by an elementary operation, then x can also be obtained from y by an elementary operation.

(b) Given any two $x, y \in D$ there is a finite sequence of elementary operations which converts x into y.

The elementary operations define a connected graph G, whose vertices are members of D and whose edges join members of D linked by an elementary operation. Different sets of elementary operations define different graphs. Sometimes, the resolution of an optimization problem is represented as a search in G for an optimal solution.

The *neighborhood* $N(x)$ of a solution $x \in D$ given by an elementary operation is the set of all vertices adjacent to x in the graph G defined by the operation.

A *local optimum* is a solution $x \in S$ such that $f(x) \geq f(y)$ for all $y \in N(x) \bigcap S$.

Let x^* be the optimal solution (or one of the optimal solutions) for the problem. A value $\zeta \in \Re$ is an *upper bound* if $\zeta \geq f(x^*)$; it is a *lower bound* if $\zeta \leq f(x^*)$.

There are two basic approaches for solving a combinatorial optimization problem: the problem can be solved either exactly – that is, an optimal solution is computed using an exact method– or approximately – a heuristic is used to compute an approximate solution. If the problem is NP-hard, then exact methods may demand an exhaustive exploration of the search space. Heuristic strategies, on the other hand, can generate a solution in a feasible amount of time, but in general provide no guarantee of finding the optimal solution.

Next we describe four meta-heuristic methods for solving optimization problems – Greedy heuristics, Hill Climbing, Simulated Annealing, and Genetic Algorithms– and an exact technique, Integer Linear Programming. Hill Climbing, Simulated Annealing and Genetic Algorithms are types of *local improvement techniques* – they start with a feasible solution for the problem and try to improve it by performing elementary operations.

Greedy Heuristics

Greedy heuristics construct a solution to a problem by following two main rules:

- At each stage of the construction of the solution, the alternatives are analyzed locally and the best choice is taken.
- Previous alternatives are not reconsidered as the heuristic progresses.

An example of a greedy heuristic is Kruskal's algorithm for computing the minimum spanning tree of a graph [37].

Greedy algorithms are in general quite fast, but they have the drawback of not being able to escape from local minima.

Hill Climbing

Hill Climbing (also called *Gradient Descent method*) is an iterative algorithm that tries to improve an existing solution by elementary operations. If an operation leads to a better solution, then the current solution is replaced by the new one; otherwise, another operation is tried. This process repeats until no further improvement is possible.

There are a number of variations of the Hill Climbing method. For example, the elementary operations can be based on random changes of the solution or on heuristic strategies. The method can also apply the first operation that causes improvement of a solution, or analyze several operations and execute the one that causes the greatest improvement.

Hill Climbing presents the same advantages and disadvantages of Greedy heuristics.

Simulated Annealing

Simulated Annealing (SA), proposed by Kirkpatrick *et al.* [107], is a type of Hill Climbing based on the principles of Statistical Mechanics [135]. The method aims to escape from local minima by applying an idea similar to the *annealing* process, in which liquids are cooled down until assuming a homogeneous form. If the cooling is sufficiently slow, then the molecular structure of the liquid has time to organize itself, resulting in the form of a crystal, which is associated with a state of minimum energy. However, if the annealing is too fast, then the system (liquid) takes an amorphic form that represents a local minimum.

The method simulates the annealing process by starting with a high temperature value $T = T_0$ and decreasing T slowly. For every temperature T, it applies a sequence of movements in order to change the state of the system. New states are obtained in the neighborhood of the current state by random elementary operations. The main rule is that the probability of the system to change from a state with energy E_1 to a new state with energy E_2 is given by:

$$p = min\{1, e^{-\frac{E2-E1}{T}}\}.$$

This function implies that the system always moves to a new state when $E_2 < E_1$. Otherwise, the new state is accepted with probability p. Figure 2.1 shows the Simulated Annealing algorithm. The value of K, the initial temperature and the rate in which T decreases, among other parameters, have to be chosen carefully.

```
1. Choose an initial configuration $\sigma$ for the system and a temperature $T=T_0$.
2. Repeat $K$ times
     (a) Choose a new configuration $\sigma'$ from the neighborhood of $\sigma$.
     (b) Let $E$ and $E'$ be the energy functions (measuring the costs) for $\sigma$ e
         $\sigma'$, respectively; if Random < $e^{(E-E')/T}$ then do $\sigma \leftarrow \sigma'$.
3. Decrease $T$.
4. If a stop condition is reached (for example, if $T$ is too small) then stop;
   otherwise, go to step 2.
```

Figure 2.1: The Simulated Annealing algorithm.

Constraints can be treated in many ways. One possibility is to consider only solutions that are feasible during the optimization processing. Another approach is to allow infeasible solutions to be generated, but penalize the non-satisfaction of a constraint in the objective function.

Simulated Annealing can be applied to solve optimization problems in general. It is expected to provide very good results, but may demand a considerable amount of runtime.

Genetic Algorithm

Genetic algorithms [81] are improvement methods based on Darwin's Theory of Natural Selection. They are characterized by a cyclic process in which a population of individuals (or chromosomes) evolve. The method consists of three basic steps, which are repeated until a stop condition is reached:

1. Selection of a subpopulation of individuals from the initial population based on their fitness (defined by a fitness function).

2. Execution of *genetic operators* on the selected individuals in order to generate a population of offsprings. The operators are usually classified as mutations and crossovers. A mutation creates a new individual by copying an existing one and changing part of its structure; a crossover operator produces new individuals by combining parts of two or more existing individuals. The mutation and crossover concepts are related to elementary operations.

3. Replacement of the previous population with the new population.

Genetic algorithms have been applied to many optimization problems, including the TSP [92, 121, 134, 183] and the Bin-packing problem [93, 126, 156, 158]. Their main advantages are the ability to explore several regions of the solution space simultaneously, and the high suitability for parallel and distributed processing. Nevertheless, genetic algorithms also demand much processing time.

Another problem, that is common for Genetic Algorithms and Simulated Annealing, is that these methods in general do not offer a guarantee about the quality of their final solutions.

Integer Linear Programming

Integer Linear Programming (ILP) is considered the most efficient general technique for solving combinatorial optimization problems in an exact way. The constraints and the objective of an optimization problem are formulated in ILP mathematically as a set of linear functions. A general form of an ILP problem with n variables and m constraints is:

$$\text{Maximize } c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1$$
$$a_{21}x_1 + a_2x_2 + \ldots + a_{2n}x_n \leq b_2$$
$$\ldots$$
$$\ldots$$
$$\ldots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m$$

$$x_1, x_2, \ldots, x_n \geq 0$$
$$x_i, i = 1, 2, \ldots, n \text{ are non-negative integers (*)},$$

where c_i, a_{ji} and b_j are constants, and x_i are variables of the problem, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$. The formulation above without the integrality constraints (*) characterizes a continuous problem that is referred to only as a *Linear Problem* (LP). This is usually solved using the simplex method [140].

Two popular methods for ILP are *Branch-and-Bound* and *Branch-and-Cut*. Branch-and-bound computes an implicit enumeration tree of LP problems. The tree starts with a single vertex containing a relaxed version of the original ILP problem without the integrality constraints (*). The method executes as follows: for each vertex P of the tree, which consists of a LP problem, the method analyzes P by computing its optimal solution X and verifying whether it is necessary to branch the tree on P. The value of the objective function $f(X)$ defines a bound for the original ILP problem. The solution X is checked for a non-integer variable x_i, $1 \leq i \leq n$. If all variables in X are integer, then the ILP problem is already solved. Otherwise, if the bound defined by P is worse than the best bound found so far, then the vertex P is removed from the tree. If none of the two previous condition are satisfied, then P is divided into two subproblems, P_1 and P_2. These problems have the same objective function and constraints of P, but include a new bound condition: P_1 is assigned a constraint $x_i \leq \lfloor v \rfloor$, and P_2 is assigned a constraint $x_i \geq \lfloor v \rfloor + 1$, where v is the value of x_i in the optimal solution for P. The problems P_1 and P_2 are then added to the tree as child vertices of P, and the branch process is repeated to another vertex not yet analyzed.

Branch-and-cut is a combination of the Branch-and-bound method with a *cutting planes algorithm*. The *cutting planes technique* consists of iteratively adding special constraints to a LP problem in order to get an integer solution. The Branch-and-Cut method tries first to solve a relaxed (LP) version of the ILP problem by using cutting planes. If an integral solution is obtained, then the problem is considered solved. Otherwise, the algorithm branches as in the Branch-and-Bound method and repeat the same process.

The downside of using the ILP technique is that the optimization problem has to be formulated as a set of linear functions. In many cases, such functions are not adequate to describe the problem. Even when they are adequate, it can be difficult to model the problem as an ILP.

2.1.3 Human-Computer Interaction and Usability Studies

Human-computer interaction has been studied for quite a while, and today helpful guidelines exist for the development of interactive systems. One of the main general aims is to build systems that are “user friendly”. Shneiderman [171] suggests, however, replacing the term “user friendly” by some clearer and more measurable human-factor goals:

1. *Time to learn* – minimizing the amount of time necessary to learn to use the system.
2. *Speed of performance* – reducing the amount of time necessary to perform a task with the system.
3. *Rate of errors by users* – minimizing the rate of errors when performing tasks.
4. *Retention over time* – improving learning, so that users still remember how to use the system after a long time without operating it.
5. *Subjective satisfaction* – increasing user’s satisfaction with the system.

These goals can be achieved by adopting principles that have been successfully applied in many systems:

1. Implementing consistent and compatible ways for the user to enter data into the system and for the system to display data.
2. Allowing frequent users to use shortcuts to speed up their main actions.
3. Reducing the *short-term memory load*, that is the amount of information that the users have to remember when performing a task with the system. This can be done by adding extra information to the interface in order to help the users.
4. Offering informative feedback for the users’ actions.

5. Preventing the users making serious errors or helping them to correct the problem without having to redo the entire work.

6. Permitting easy reversal of actions.

There are several types of systems regarding interaction styles. Shneiderman [171] discusses some examples of styles: menu selection, form filling, command-language based approaches, natural language and direct manipulation. Here we focus on direct manipulation.

Direct manipulation implements a visual representation of the problem to be solved by the user, so that he or she can perform tasks by directly manipulating objects. The main features of this approach are: (1) a continuous representation of the objects and actions of interest; and (2) the use of physical actions such as clicking on buttons and dragging and dropping objects on the screen, instead of using complex syntax commands. Furthermore, direct manipulation allows incremental reversible operations that have an immediate visible effect on the object of interest. Examples of Direct-Manipulation systems include text editors, electronic spreadsheets, computer-aided design (CAD) systems, games and geographic information systems.

A benefit in using direct manipulation pointed out by Shneiderman is that novice users can learn basic functions quickly by playing with the system, or through a demonstration by a more experienced user. Intermittent and frequent users also benefit from the approach as it makes operational concepts easy to remember and allows fast execution of tasks. Moreover, users receive immediate feedback of their actions, and can change the direction of their activity if these results are not positive.

There are, however, problems with the direct manipulation approach, which are related to the use of a visual representation:

- The representation may be too large and take more than the visible screen space to be properly displayed. Therefore, repetitive and annoying scrolling of the visualization may be necessary.

- The visual representation may be meaningful for the designer of the system, but not to the final user.

- The representation may also be misleading; for example, the user may understand its general meaning, but may incorrectly interpret how to interact with it.

- Direct interaction with the visual representation may not be as efficient as typing a command for some particular problems.

Despite these problems, direct manipulation still appears as a more natural and easy-to-remember interaction style than other approaches such as menu selection and command typing. This is the case for most of the interactive systems discussed in the next sections. Direct manipulation can also be combined with other approaches when good visualizations for some objects of interest cannot be found.

All interactive systems involve tasks that must be performed by the user, as well as tasks that are to be executed by the computer. An important issue for the designer of such systems is to find a good balance between automation and human control. Shneiderman recommends simplifying the user's role by eliminating human action when no judgment is required, and avoiding repetitive, tedious, and error-prone tasks. Instead, the user should concentrate on creative tasks, critical decisions, strategic planning, and on coping with unexpected situations. The computer, on the other hand, should be used to manipulate large volumes of data, to execute repetitive preprogrammed action reliably, to monitor and control well pre-specified events, and to execute complex mathematical and logical operations.

The design of an interactive system must be tested to verify whether it attends the human-factor goals specified previously. The system can be tested not only after implementation, but also in much earlier stages of its development in order to aid with the choice and validation of design solutions. Tichy [184] comments that many Computer Science papers (including publications in Software Engineering) have unsupported claims; he proposes that computer scientists should do more experimentation.

Several approaches exist for evaluating a system; examples are pilot studies of design solutions and rapid prototyping. System evaluation can be done through interviews and group discussions, surveys, and controlled human experiments.

Evaluation based on human experiments follows methodologies from psychology. The basic steps for this type of experiments involve stating a testable hypothesis, building a well controlled setup, measuring the aspects of interest with a significant number of subjects, and analyzing the results using statistics.

Sometimes large human experiments are not possible because the application depends on expert domain knowledge, and expert users are unwilling or not available in a sufficient number. Nielsen [144] suggests an evaluation technique for this situation called *heuristic evaluation*.

Heuristic evaluation is a systematic inspection of a user interface design for usability problems. The evaluation is performed by a small number of evaluators who examine the interface and try to identify good and bad aspects of it, according to a list of recognized usability principles. Each individual evaluator inspects the interface separately; only after that are they allowed to communicate and aggregate their findings. The best number of evaluators depends on the application, but it has been recommended to use about five and at least three persons. An individual evaluation section usually takes between one to two hours. The results of the examinations can be recorded in written reports by each evaluator, or by an observer who is present in the sessions and annotates the comments vocalized by the evaluators. The observer can also assist with the operation of the system (this may be necessary when the system is not fully implemented, or when the interface is too complex and the evaluator has no time to be trained to use it). Furthermore, the observer must answer evaluators' questions about the functioning of the system.

Some studies have shown that domain experts identify more usability problems than novice users. Moreover, evaluators that are not only domain experts but are also experienced with the evaluation of systems have provided much better feedback.

2.1.4 Information Visualization

Information Visualization is an emerging area that studies ways of amplifying cognition of ideas, processes, and phenomena through visual representations. Techniques from this area can be used either to communicate information or to create and discover it.

There are many techniques for Information Visualization and we do not intend to cover them in this thesis. For a good survey of the field, we recommend the books of Card *et al.* [31] and Spence [174]. We only illustrate a few approaches in order to show how broad and fascinating this area is. Most information visualization techniques explore visual aspects such as dimensionality, color, object size and shape, and animation or dynamic changes of visual representation. Some ideas involve concepts of direct manipulation, where users can rotate, move or select objects on the screen in order to gain a better understanding of a phenomenon or concept.

A typical example of dimensionality is a 3D visualization of a mathematical function based on two variables (see Figure 2.2). This visualization supports understanding of mathematical concepts such as continuity and minimum value. The user may also change the view point in order to see regions of the graphics that are currently occluded.

Color and other graphical properties such as size and shape can be employed to classify objects and to emphasize information that requires immediate attention of the user. Figure 2.3 shows an example where color is used to discriminate regions and periods of the year where bush fires are

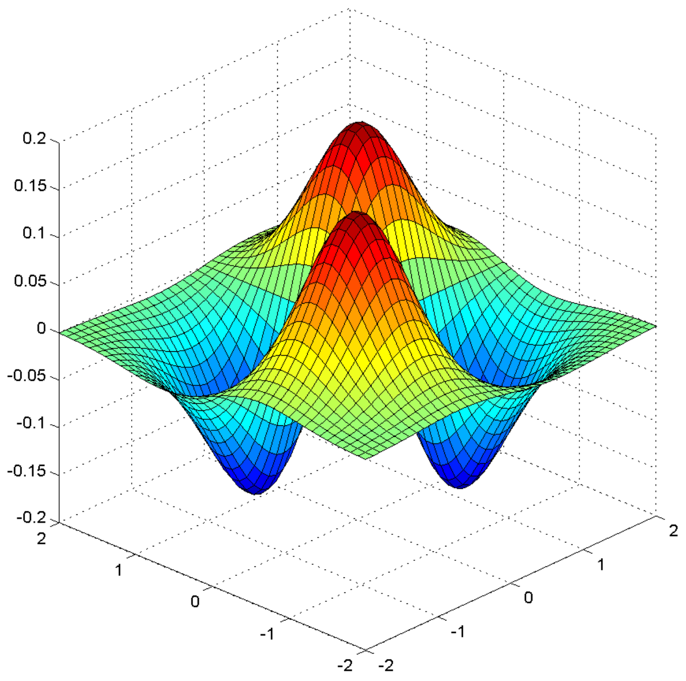


Figure 2.2: 3D plot of a mathematical function, created using MATLAB – Mathworks Inc.

more common in Australia.

Another interesting visualization technique is given by a simple combination of vertical lines in a structure called *Parallel Coordinates* (see illustration in Figure 2.4). Parallel Coordinates are a good approach for interactively searching for objects with many attributes.

These are just a few examples of techniques for Information Visualization. Another approach, that is also very popular, consists of modeling the object or problem of interest as a graph, and then producing a drawing of the graph. This approach has been studied extensively in an independent area named *Graph Drawing*. We discuss Graph Drawing in more details in Chapter 5.

2.2 Human Interaction in Related Areas

This section presents the breadth of activities in human-computer collaboration, by describing examples of its applications for three general areas.

We start with the area of Computer Aided Design, where we introduce categories of interactive paradigms for graphical-object modeling. Examples of interactive systems that fall into some of these categories are presented.

Next we describe systems that exploit human interaction in Information Retrieval. Users are already an essential element of information retrieval processes. We show, however, that user feedback

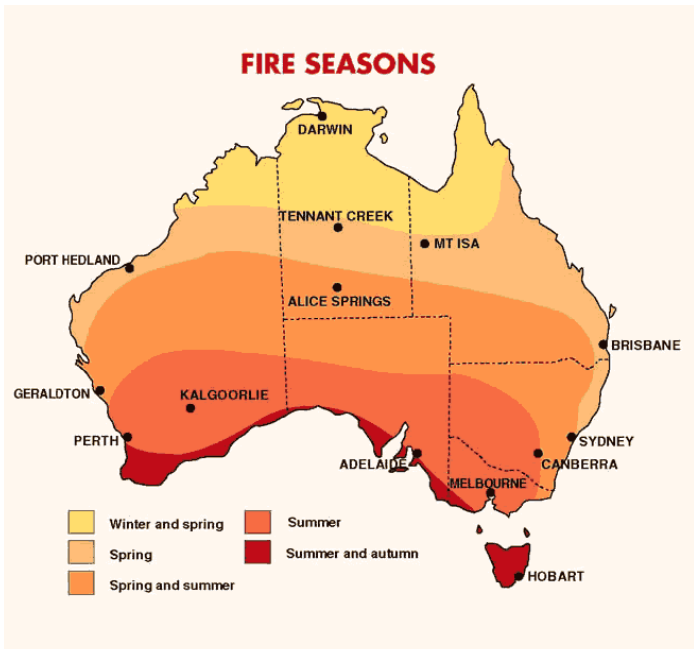


Figure 2.3: A colored map indicating fire seasons in Australia. Picture from the Australian Commonwealth Bureau of Meteorology. Used with permission. *http://www.bom.gov.au/inside/services_policy/fire_ag/bushfire/threat.htm*.

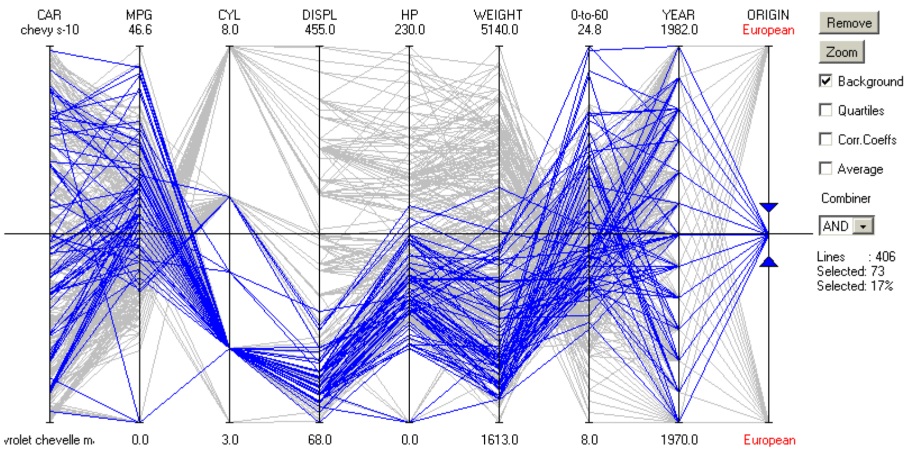


Figure 2.4: An example of use of Parallel Coordinates for visualizing characteristics of several cars such as model, number of cylinders, weight, and year and place of fabrication. Attributes are described on the vertical axis, so that aspects referring to the same car are connected by lines. The visualization was filtered interactively for emphasizing characteristics of cars produced in Europe. (Picture obtained with the Java Applet at *http://www.cs.uta.fi/˜hs/pce/*. Courtesy of Harri Siirtola.)

can also be exploited to provide more effective search and filtering services.

Finally, we present an approach called “Mixed Initiatives”, that offers principles for combining user actions with tasks performed by “intelligent agents”. This approach has been successfully applied for the development of effective user interfaces.

The systems described in this section gives us an idea of how broad the field of human-computer collaboration can be. The concepts and principles introduced are also useful, since they provide directions for the development of interactive systems for combinatorial optimization.

2.2.1 Computer Aided Design

Kochhar, Marks and Friedell [114] classify paradigms for graphical-object modeling into six categories. These categories represent different degrees of automation in which design decisions are taken and implementation details for the design are defined. Such classification, even though proposed for graphical modeling, is also useful in this thesis since it helps to distinguish between several other types of interactive systems. The categories provided by Kochhar *et al.* ranges from completely manual to completely automated paradigms. In the latter cases, the degree of automation is more flexible, as the user can dynamically adjust the system to have a higher or lower control of the design process. The categories of interactive paradigms are listed below:

1. Fully Manual – The user has control over the entire modeling process and is responsible for all design decisions. The system implements support for mainly low-level operations.

2. Constraint-based – the modeling is based on constraint satisfaction with the system satisfying constraints related to the domain knowledge of the application. The user is responsible for all design decisions, but his or her actions are limited by the system so that constraints are not violated. A typical example of the constraint-based paradigm is an application for interior design: the user defines the layout of objects representing furniture by moving them around on the screen; the user movements are, however, constrained by the system in order to prevent overlapping of objects. The Constraint-based paradigm can be exploited in another way: the system starts with little or no information about the design at all, and tries to satisfy constraints that are entered incrementally by the user. The modeling task happens with the user defining constraints that describe the characteristics of the design, while the system produces a design solution that satisfies all user constraints. An example of such system is described later in this section.

3. Critic-Based – The design is still done manually by the user; however, the system helps the modeling task by identifying portions of the graphical objects that may need improvement. Critic agents are directly invoked by the user or can be automatically activated by the system to provide criticisms about constraint violation or low-quality aspects of the current design.

4. Improver-Based – this is a step further to the critic-based paradigm. In this case, the system is not only capable of identifying flaws in the user-generated design, but can also improve it. An example of an improver-based system is described in this section.

5. Fully Automated – In a fully manual modeling, the system is responsible completely for the overall design or for the details of the design solution. The user is in general passive in this approach.

6. Cooperative CAD (CCAD) – This is new type of paradigm proposed by Kochar and others. In a CCAD approach the user provides a partial design and lets the system create alternative solutions for certain open portions of the design. The user can then browse the solutions, and choose one that better matches his or her expectations. The design can be refined continuously by having the user perform manual adjustment of the computer-generated solution, or running the system again to produce alternative solutions for other design aspects. The CCAD paradigm allows a dynamic adjustment of the degree of automation, as the system can be used in either a fully manual approach or a fully automated approach. An example of the CCAD paradigm is the *Design Gallery* methodology, which is also described at the end of this section.

Next, we present three examples of interactive systems and approaches that fit in the categories above.

A Drawing Beautifier

Bolz [24] introduces a *beautifier* for drawings that is integrated into a graphical editor. The beautifier is an improver-based subsystem that implements an extensible knowledge base about graphical problems. It inputs sketches produced by the user and solves problems such as gaps, oblique lines, and misalignments. Figure 2.5 illustrates the processing executed by the system described by Bolz. Figure 2.5(a) is an initial drawing and Figure 2.5(b) is its improved version, after the beautification process. The system offers several options to the user: the beautifier can be activated manually or

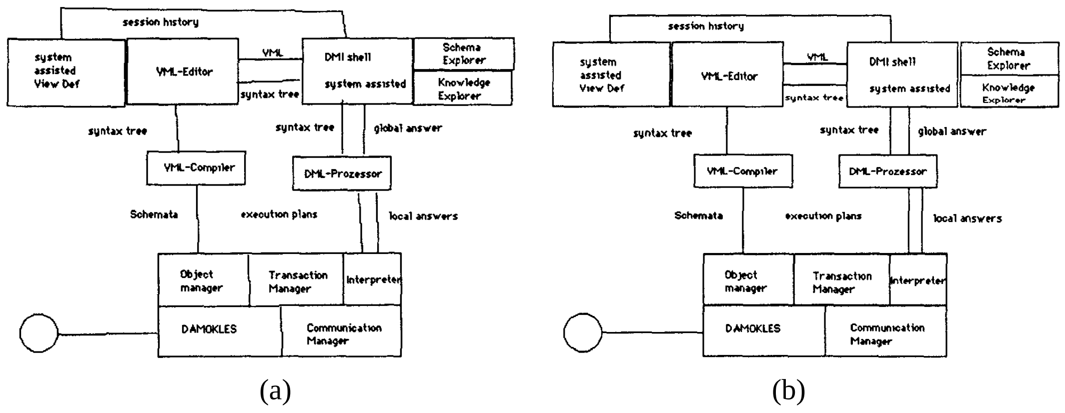
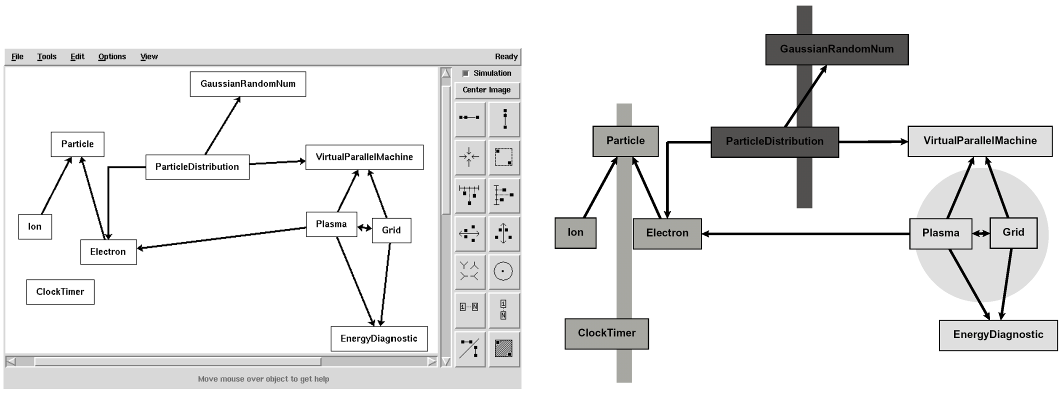


Figure 2.5: The beautification of a drawing. Figure (a) is the initial drawing and (b) is the improved version after the beautification. Courtesy of Dieter Bolz.

can be set to run automatically after a period of time. The user can undo and redo improvements executed by the beautifier, as well as concentrate the beatification process on a particular region of the drawing. Moreover, the user can set and adjust the parameters that control the beautifier (for example, the maximal angle between two oblique lines that characterizes a graphical problem) dynamically. If the user desires, he or she can request the system to automatically estimate the parameters for the beautifier based on an analysis of the current drawing. The system is also capable of showing both the previous and the new drawing simultaneously in an overlapping way. This visualization facilitates the changes made by the beautifier. A similar beautification system is presented in [196].

Constrained Graph Drawing Using Springs

Ryall, Marks and Shieber [165, 166] present an interactive constraint-based graph drawing system called *GLIDE* that uses force-directed placement based on the spring algorithm [57]. The system works as follows: a graph is modeled as an energy system composed of springs connecting every pair of vertices; a method for iterative quadratic optimization is set to continuously compute a layout that corresponds to a state of minimal energy. While the optimization method is running, the user can adjust the drawing according to his or her desires by adding constraints to the model. Constraints in this system are called *VOFs* (*Visual Organization Features*). They include a variety of layout criteria such as: showing two vertices close to each other, showing an edge as an orthogonal line, constraining a group of vertices to have the same horizontal or vertical coordinate, etc. VOFs are implemented within the system as extra springs that are added to the original energy model. The system solves theses constraints by searching for a new state that minimizes the energy of



(a) (b)

Figure 2.6: GLIDE: an interactive constraint-based system for drawing graphs [165]. Figure (a) is a snapshot of the system with an initial graph drawing. Figure (b) shows the VOFs been applied to the drawing. Courtesy of Joe Marks.

the entire set of springs (the original springs of the graph plus the constraint springs). The user may move vertices manually during runtime to help the system to escape from configurations that represent local minimal. Figure 2.6 shows a graph drawing and VOF constraints in *GLIDE*. VOFs are represented by special graphical objects such as bars and circles on the screen.

Design Galleries

Marks *et al.* [129] introduce *Design Gallery*, an approach for designing computer graphics and animations that follows the model of a CCAD system. The approach is based on a good balance between design activities executed by the computer and the user. The computer is responsible for two processes: *dispersion* and *arrangement*. In the dispersion process, a spectrum of representative design images or animations are automatically generated by varying a set of tuning parameters. This can be very time consuming and is usually executed off-line. During the arrangement process, the generated solutions via dispersion are organized according to some distance metric, and are presented to the user in the form of a graphical gallery. Figure 2.7 shows two examples of Design Gallery interfaces – for light selection and placement, and for designing animations of a particle system. The interfaces display thumbnails of several design solutions obtained by the dispersion process. The thumbnails are arranged so that solutions with similar properties are shown close to each other in the gallery as much as possible. Large views of the solutions can be obtained by clicking on them. The task performed by the user in this approach is simplified to focusing the parameters for exploration, and choosing the solutions from the gallery that they like most. Design Gallery was also applied to the problem of drawing general graphs [7]. Figure 2.8 shows a snapshot

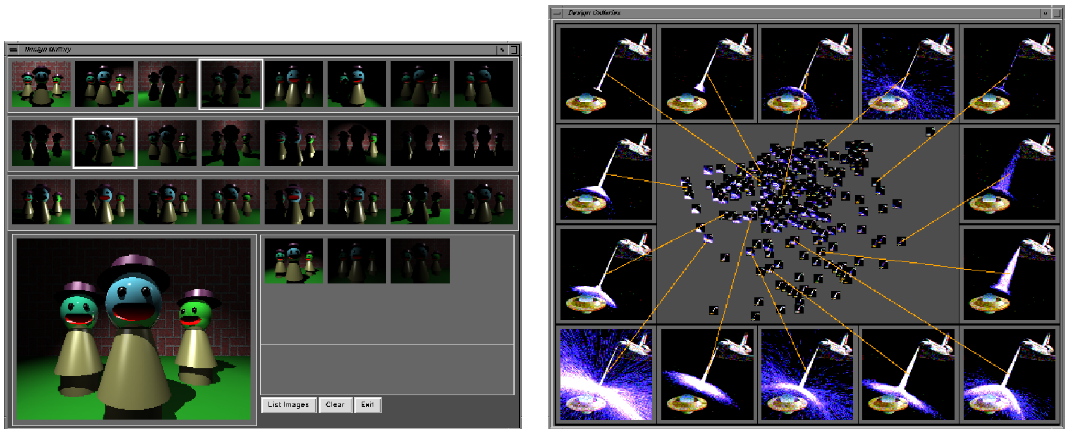

(a) (b)

Figure 2.7: Design Gallery interfaces for (a) light selection and placement and for (b) particles system animation [129]. Courtesy of Joe Marks.

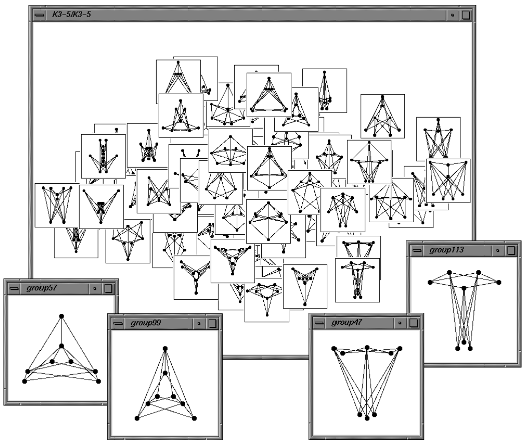


Figure 2.8: SMILE: a Design Gallery system for Graph Drawing [7]. Courtesy of Joe Marks.

of *SMILE*, a Design Gallery system for Graph Drawing.

In the following sections we present examples of interactive systems and approaches to other problems. Some of the examples can be classified as belonging to one of the categories described previously. However, most commonly they will share attributes of two or more categories.

Not all configurations of interactive systems, however, are relevant to this thesis. In the informative article “Machines that learn from hints” [3], for example, Abu-Mostafa describes applications of neural networks to real-world problems. The term “hints” is used to refer to representative input data that is chosen by the user in order to train a neural network. In another article, “Exploring component-based representations – the secret of creativity by evolution” [17], Bentley shows how computational evolution can be used to generate creative designs for problems that appear in Architecture and Engineering. The approach consists of having the user define the basic rules and properties that describe a physical object; an evolutionary algorithm is then used to build the object by exploring alternative design solutions. An example is given where a system creates paper objects that fall as slowly as possible. Even though interesting, both approaches are usually based on a sequential process, where human interaction happens only in the initial stage for setting up the system and or for inputting data. After that, the system runs automatically, without human interference for a long period of time. Such systems are not within the scope of this thesis[2].

2.2.2 Systems for Information Retrieval

Paraphrasing Ingwersen [98]: ”*Information Retrieval* covers problems relating to the effective storage, access, and searching of information required by individuals”. In Information Retrieval applications, a system usually has to recover all relevant information from a database that matches a query defined by a user. This is necessary for example when searching books and magazines in libraries, and for finding material on the Internet using Web search engines. Human interaction is an essential part of the process of retrieving information. Interaction allows the user to review the search process in order to reduce the number of retrieved documents, or to include information that is relevant but that was not covered by the initial query. Human interaction in this domain can be done using a few techniques such as defining a list of keywords for the search, and possibly combining keywords using boolean rules. The search mechanisms may retrieve all information that matches exactly the user query, or the best-matching information. The latter approach is considered to be more effective. Another form of interaction that has been shown to be effective is *Relevance Feedback*. In this approach the user marks some documents as being relevant; the system then expands the query using this information. The relevance feedback is given either explicitly – with the user directly marking the documents– or implicitly – with the system assuming relevance based on some operations done by the users such as viewing the whole document or printing it. The expansion of the query can also be done automatically or manually. In an automatic approach the system includes (or removes) keywords from the query based on the content of the relevant documents. In a manual process, keywords recovered from the relevant documents are presented to the user, who decide whether they should be included in the query.

Koenemann and Belkin [98] have done research to determine how a relevance feedback component impacts the information seeking behavior and effectiveness of novice searchers in an in-

[2]Bentley also refers to some interactive systems where humans guide the evolutionary process by playing the role of a fitness function. This approach is discussed in Section 2.3.1

teractive environment. They developed four versions of an interactive system where users have a gradually increasing control of a relevant feedback tool. In the simplest version of the system, users can only perform searches with keywords; no relevance feedback is available. In the most complex version, the users can select some documents as relevant and decide as to how these documents should affect the query for the next search. Experiments with the systems showed that users clearly benefited from the opportunity to revise queries in an interactive process. Moreover, users in the most interactive version of the system need fewer iterations to achieve results comparable to, or better than the other less interactive conditions. These conclusions suggested that interfaces for Information Retrieval should be designed to support interactive collaboration between the users and search engines for formulation and reformulation of search queries.

Another interactive problem that comes from a subarea of Information Retrieval is called *Information Filtering*, which also deals with the selection of information that matches some criteria specified by the user. Information filtering is necessary, for example, for detecting important incoming messages that should be seen immediately by the readers or, on the other extreme, for filtering and deleting undesirable e-mails such as spam and hoaxes. Human interaction in filtering systems borrow the same techniques from Information Retrieval. Basically, the users can specify keywords and rules that describe the characteristics of messages expected to be filtered. Another interaction is to have the users rating a few messages, and let the system try to predict the rate of every unread article by comparing them with the rated samples.

Kilander *et al.* [106] investigated techniques for filtering electronic messages in a project called *IntFilter*. The project was restricted to deal with messages from the Internet Usenet News, and involved the development of several prototype systems for testing filtering techniques. Their last prototype, called PEFNA (*Private Filtering News Agent*), allowed users to label messages that they considered relevant. The labels were names describing user-defined categories. After an initial labeling, the filter in PEFNA measured the distance between each unread article and the categories, by comparing words in the article with the content of the labeled messages. These measurements were then used to sort the articles, so that the readers could visualize which ones are more related to the defined categories (such articles would appear at the top of a sorted list for a chosen category).

The IntFilter project was discontinued in 1997 without the implementation of many planned features. However, Kilander and the other researches presented in their final work report [106] a few ideas that emerged from the experiments with the prototype systems. They realized that making automatic filters to match the human knowledge in textual analysis is quite difficult. Natural

Language is complex and a simple word comparison (which does not consider the semantic of the text) is not capable of correctly classifying all articles. On the other hand, the researchers perceived that ordinary people are good at making intuitive decisions based on whatever information is available. They then suggested that computer power and human skill should be exploited in a more appropriated way. Rather than having the system automatically delete messages that are considered irrelevant by the filter or perform actions based on a possibly important message, the users should still be responsible for these decisions. The roles of the system should be to sort and display information about the messages, in order to provide the users with sufficient clues to support their decision-making process. As a conclusion, Kilander and his colleagues proposed that new visualizations should be investigated in future research. The aim would be to identify good visualizations, which display the main properties of categories of texts and help readers to distinguish between them.

2.2.3 Mixed Initiative Systems

The *Mixed-Initiative* approach [90] combines the capabilities of several agents so that each agent can contribute with the task that it performs best to solving a problem. It is possible to have agents consisting only of automated processes. However, the usual setup is a combination of intelligent automated services provided by computer agents with direct manipulation done by human agents. The main characteristic of Mixed Initiative is that the *initiative* (that is, the control of the interaction) changes between agents over time; while an agent is having has the initiative, the others assist it. The agents may also work independently and assist each other when asked. Moreover, the agents may dynamically adjust their interaction style and roles when necessary, to best address the problem. The interaction between agents is described as a dialogue. Natural language dialogs are recommended when one of the agents is a human.

Mixed Initiative interaction has been applied to planning and scheduling problems [67, 94, 173].

Horvitz [94] presents principles for effective integration of automated services with direct manipulation interfaces in Mixed-Initiative systems. Some of these principles are:

- Providing automated services that effectively contribute to achieve the user's goals.
- Considering uncertainty about the user's goals and exploiting it.
- Employing dialog to resolve key uncertainties.

- Considering the cost and benefits of performing an automated action based on the status of the user's attention, and deciding about the best time to execute the action.
- Allowing the user to directly invoke and terminate an automatic action.
- Including automatic tools that can be used by the user to refine automatically-generated results.
- Maintaining working memory of recent interactions.
- Allowing the system to continuously improve its effectiveness by observing the user and learning the user's goals and needs.

An example of a Mixed-Initiative interface is the LookOut project [94], which implements automated scheduling services on Microsoft Outlook. LookOut offers semi-automatic services that help to create appointments based on email messages. The system parses an email message and identifies patterns that suggest important events. Calendar appointments are then automatically proposed for these events, according to an estimation about the users goals and needs. The system learns by working with the users, so that the type of messages that are filtered and the level of automation can be dynamically adjusted. A strong emphasis is put on a natural dialog between the user and the system, which occurs via dialog boxes with textual information, a graphical agent, voice messages and speech recognition.

2.3 Interactive Optimization

The interactive systems and approaches described in the previous section deal with problems in which combinatorial optimization processes could be present, but they were not the main focus of attention. In the present section, we investigate approaches which explicitly combine human interaction with optimization algorithms in order to solve combinatorial optimization problems. Several examples of interactive systems for combinatorial optimization are presented.

We first describe some "old" systems – from 1988 and 1995 — that already indicated that human-computer interaction could be a promising way to address the difficulties of real-world optimization processes.

Arnold and Scott [9] presented in 1988 an interactive maze router for a VLSI layout editor. The router allows *graphical hints* specified by the users. Two types of hints are available: *fences* and *magnets*. Fences are boundary areas of the layout that cannot be crossed. They designate regions inside which a route has to stay or define areas that must not be invaded. Magnets are objects that attract routes. Human-computer collaboration in this system happens as follows: the user specifies hints to suggest a path to the router. The system then automatically generates a path with minimum length that does not cross any fence and passes as near to the magnets as possible. Since fences restrict the possibilities of routing the path, one of its side-effects is to reduce processing time necessary to compute a high quality solution.

Bachmann [10] and Költze [115] designed interactive systems for scheduling city services in the German city of Passau. These systems implement algorithms for testing the feasibility of interactive user decisions and computing assignments by a weighted greedy heuristic on a selected portion of the data. The user interacts with the systems by selecting a subset of the not yet scheduled data for optimization, and making assignments by hand. The user's assignments receive highest priority and are not overruled or undone by any algorithm.

One of the systems was used by Professor Franz J. Brandenburg [25] to reduce the number of bus drivers needed. The schedules produced interactively were better than previous handmade assignments and better than results obtained by fully automatic algorithms.

It is possible, even probable, that other similar systems were developed (and continue to be designed) in-house by some industries. Unfortunately, it seems that there was no interest in generalizing the main concepts in those applications. In fact, only recently – in the last five years – the concepts of human-computer interaction for optimization problems have been investigated by researchers in a systematic way. Many studies have been done in parallel with our User Hints research by independent groups. These studies revive familiar terms such as "human-in-the-loop" and "user control", or propose new expressions, such as "human guidance", to describe the role that a user can play in an interactive environment. Despite the differences in terminology, they have the common aim of providing a more systematic way for integrating human skills with automatic methods in combinatorial optimization processes; our approach has the same goal.

Next we describe examples of related studies. They share many features with our User Hints framework.

2.3.1 Interactive Evolutionary Approaches

Interactive optimization using evolutionary algorithms (such as Genetic Algorithms) has been investigated recently. The most common approach is to have the user performing the role of a fitness

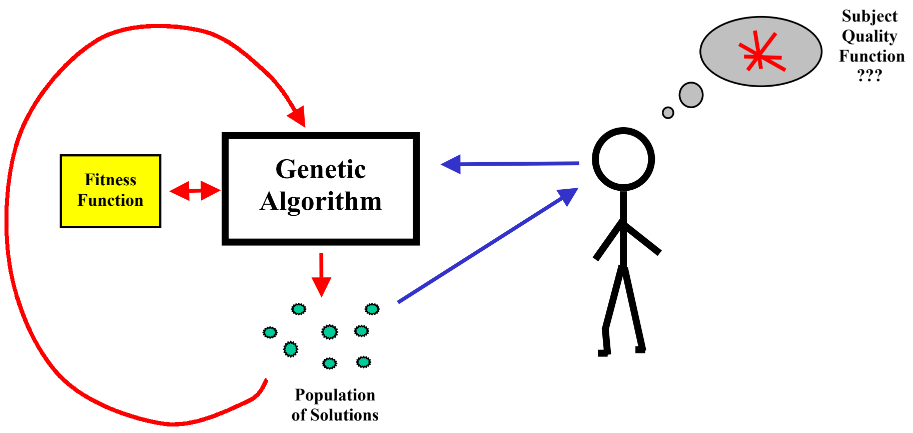


Figure 2.9: Interactive Evolutionary Computation where the user performs the fitness function.

function. This is illustrated in Figure 2.9.

Rosete-Suárez *et al.* [163], Jacobsen [99], and Barbosa *et al.* [12] present systems for drawing graphs that learn the user's aesthetic criteria. These systems use genetic algorithms to generate a population of drawings based on a dynamic multi-objective fitness function. Aesthetic criteria for graph drawing can be highly subjective and difficult for the user to describe precisely [40]. A solution for such a situation is to allow the user to interact with the genetic algorithm in order to tune the fitness function in an intuitive way. Instead of applying the genetic algorithm to a specific task, the user only indicates whether it is on the right path by providing an additional evaluation of the computer-generated drawings. The system described by Jacobsen, for example, tries to minimize a weighted function of seven aesthetic criteria: showing few edge crossings, presenting high angular resolution between lines, and displaying as many symmetries as possible (these aesthetic criteria are common in many approaches for drawing graphs [43]). The genetic algorithm runs for a small number of iterations and gets feedback from the user. Basically, it displays the best eight drawings currently produced – one drawing for each one of the seven aesthetic criteria, and the best drawing according to the total weighted function–, and the user provides scores between 0 to 9 for each drawing. The system then uses the scores to adjust the importance of the aesthetic criteria in the weighted fitness function. For instance, giving high scores to drawings with few edge crossings increases the weight of the edge crossing criterion in the fitness function. The process repeats until the genetic algorithm produces a drawing whose quality matches the user's expectations. Note that the main goal is to create a desirable drawing, rather than learning a general fitness function that can be employed for new graph drawing problems. The fitness function computed during the interactive process may or not be suitable for other graphs[3].

One advantage in using genetic algorithms for interactive optimization is that it naturally produces several alternative solutions for evaluation. Moreover, genetic algorithms are flexible in handling a considerable number of different objectives and constraints. A survey of interactive evolutionary algorithms that follow the approach described above is presented by Takagi [180].

2.3.2 Haptic Hints

Bayazit *et al.* [13, 14] present an interactive framework for motion planning that aggregates elements of human-computer collaboration, haptic devices and visual representations. The general motion planning problem consists of finding a sequence of configurations – that we refer here to as a *path* – that takes an object form an initial state (or location) s_0 to a final state (or location) s_f, while avoiding collision with obstacles. The object can be any element such as a robot or a robot arm. The problem is complex when there is a large number of configurations to explore, and when the path depends on some 'critical' configurations that are difficult to find. On the other hand, solutions for these cases may be intuitive for a user given that a visual representation is provided.

The framework exploits the user's knowledge by displaying a 3D representation of the object and the environment in which it is inserted. The user can manually move the object by using a PHANTOM haptic device [131]. Force feedback is employed to indicate collision between the object and obstacles.

The role of the user in the framework is to provide 'haptic hints', that is, a complete path or partial paths for critical situations, by using the haptic device and the 3D representation. The paths may contain collision, however, they describe an approximate solution for the problem. The computer's role is to take the user-created paths and to connect and improve them in order to produce a complete collision-free path.

Figures 2.10(a) and (b) present examples of motion planning problems investigated by Amato *et al.*. Figure 2.10(b) is the *flange* problem, in which the aim is to insert a curved pipe into a circular opening of a fixed object. The picture shows a configuration extracted from a user-generated path.

[3]Mendonça [40] investigates learning a fitness function and some control parameters of a Simulated Annealing for future use. The learning process happens by having the user improve a drawing and inputting it to a general Simulated Annealing; the general Simulated Annealing extracts the quality attributes of the drawing and uses them for adjusting a weighted objective function and for tuning the control parameters. Later the user can call another Simulated Annealing that employs the learned elements for improving a handmade drawing. Masui [133] presents a more elaborate learning process that applies genetic programming for evolving complex objective functions; this approach is based on samples of good and bad drawings previously entered by the user, and is much closer to an automatic model than to an interactive approach.

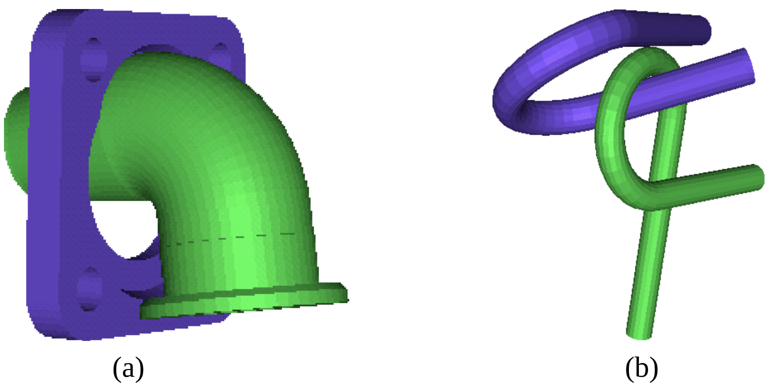


Figure 2.10: Examples of problems for the Haptic Hints framework. The *flange* problem (a) and the *alpha puzzle* problem (b) from [13]. Courtesy of Nancy Amato and Osman Bayazit.

In the configuration the pipe collides with the rectangular object; however, the system can improve this solution by pushing the pipe down. Figure 2.10(a) is a more difficult problem – the *alpha puzzle* problem – where the two twisted tubes have to be separated (one tube is fixed). The figure shows a configuration with a narrow passage found by the system by improving a user-generated solution.

Experiments with the framework showed that haptic hints allowed more complex problems to be solved and in a much faster way than could be done by a fully automatic motion planning method.

The Haptic Hints framework was designed for motion planning problems that involve a 3D modeling of the environment. We believe that the framework could possibly be extended to deal with problems on a 2D representation whose solutions can be represented by a path. The haptic device may be useful to provide a sense of constraint violation as feedback to the user actions. Note that the work of Amato has similarities with the problem treated by Arnold and Scott in their maze router for VLSI layout (discussed at the beginning of Section 2.3). Despite some differences, both approaches deal with the problem of computing a path and use interactive techniques to suggest an approximate path to an automatic method.

2.3.3 Human-Guided Search

A general approach for human interaction is the cooperative paradigm of *Human-Guided Search* (HuGS) [8, 110, 170]. This paradigm is quite similar to our User Hints framework, and was developed about the same time. We describe it in detail.

The first version of HuGS was called *Human-Guided Simple Search* (or HuGSS) and was introduced by Anderson *et al.* [8]. It was considered the first appearance of a general interactive approach for solving optimization problems that aggregates many ideas simultaneously such as having the human perform an active role, focus of optimization methods, manual changes, and controlling of the execution time of a search method. The HuGSS paradigm divided the optimization process into two main subtasks carried out by different entities. The computer was responsible for finding local minima using a simple Hill Climbing search, while the user worked on escaping from local minima and leading the search towards better solutions. The approach was successfully applied to the *capacitated-vehicle-routing-with-time-windows* (CVRTW) problem. In simple terms, this problem consists of defining routes that leave from a central depot, pass by customers at fixed geographical locations and return to the origin point. Each route is assigned to a truck; moreover, each customer has to be served by exactly one truck, and the service has to be executed within a time window. The optimization problem is to compute the minimum number of routes that serve all customers according to their time constraints, and present minimum total length. Anderson and others developed a system for the CVRTW problem where the user can perform three main interactive actions: manually changing an existing solution; focusing two Hill Climbing algorithms (based on exhaustive search) on a particular area of the solution; and reverting to an earlier solution. The focus of action consists of setting search priorities for each customer and/or defining how deeply the automatic search should be executed. Three levels of priority – low, medium and high – are defined, so that only high-priority customers can be reassigned by the search algorithms to a different route, and this movement has to be done to routes that have no low-priority customers. A visualization of the solution under improvement provides immediate feedback to the user. Picture 2.11 shows a snapshot of the system.

Human experiments with the interactive system were carried out and showed that human guidance using simple local-improvement methods could provide much better solutions than an unguided search. Moreover, the results were comparable to the ones produced by state-of-the-art methods for the CVRTW problem. Another advantage that was realized was that their interactive approach could be used to manage infeasible instances of the problem quite well, without needing to change the algorithms implemented in the system. The experiments involved a special table with a large screen area. This table and the ones that we built for our project are described in the Appendix A.

The HuGSS paradigm was applied to other optimization problems, such as Job Shop scheduling and Graph Partitioning, with the development of interactive systems. For the Graph Partitioning problem[4], for example, a system was implemented where users could help automatic methods to

[4]A definition of the Graph Partitioning (or Graph Clustering) problem is given in Chapter 4.

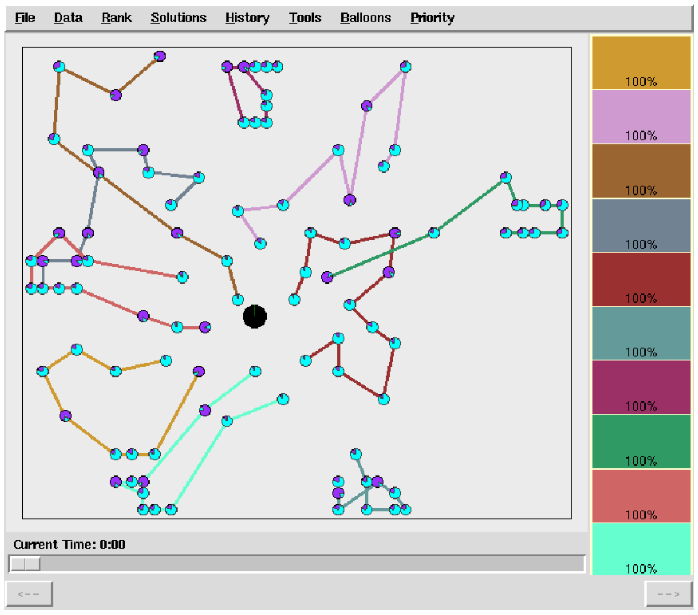


Figure 2.11: The HuGSS system for the Capacitated-Vehicle-Routing-with-Time-Windows problem [8]. Courtesy of Joe Marks.

improve graph partitioning solutions [124]. Among the interactive actions, the users could manually move vertices between partitions, choose an appropriate partitioning method to run, and focus the method on a portion of the graph.

A more detailed investigation of the HuGSS paradigm was done later for the CVRTW problem, where the researchers analyzed several aspects regarding human-computer collaboration [170]. The investigation showed that the focus mechanism did contribute to improving the quality of the solutions. It was also noted that users had different strategies for working with the system. This suggested that using a collaborative approach, involving a group of people, could lead to better results. Two other interesting findings of the research were that users could focus the search on effectively promising regions of the problem, and could stop the execution of search algorithms when no significant improvement could be obtained.

In 2002, the original HuGSS paradigm was extended by Klau *et al.* [109] to include a tabu-search method. Experiments were conducted for four optimization problems, to know: (1) the edge-crossing minimization problem for drawing layered graphs, (2) a variant of the TSP problem where there is no requirement to visit every location, (3) a simplified version of the protein-folding problem, and (4) a Jobshop scheduling problem. Experiments using the paradigm demonstrated that a human-guided tabu search could produce better results than an unguided version of the same method (with the tabu search running on the entire problem for a considerable amount of time without human interference). The experiments also showed that the guided tabu search outperformed the simple guided Hill Climbing methods (based on an exhaustive search) used in the first HuGSS paradigm.

Since the results with the tabu search were promising, the extended approach was renamed Human-Guided Search framework (HuGS) [110] – without the term "Simple", which referred to a simple Hill Climbing search. A toolkit for using HuGS for other optimization problems is presented in [110].

The new HuGS framework includes the original Hill Climbing methods and the tabu search. The main interactive actions that can be performed by the users are:

- Manual changes of the current solution.
- Invoking, monitoring, and halting a search for a better solution.
- Focusing the search by defining three levels of mobility (low, medium and high) for the elements of the problem. The search is aimed to change the attributes of high-mobility elements; medium-mobility elements can only be changed to service changes of high-mobility ones; low-mobility elements are never modified.
- Reverting to a previous or pre-computed solution.

2.3.4 Other Approaches

In this section we present a few other examples of interactive approaches. They are restricted to more specific problem domains.

Louis and Tang [127] present an interactive divide-and-conquer approach for the Traveling Salesman Problem using a genetic algorithm. In their approach, called *IGA* (*Interactive Genetic Algorithm*), the user divides the cities of a TSP into smaller disjoint clusters. This operation is executed visually with a graphical interface, as shown in Figure 2.12(a). The system then considers each cluster a separate TSP problem, and solves it independently by running a genetic algorithm. The solutions produced for the independent TSPs – that is, a tour for each cluster – are displayed on the screen (see Figure 2.12(b)). Finally, the user combines the sub-tours in order to create a solution for the entire set of cities. The combination can be performed either manually or by a semi-automatic process. In the manual approach, the user combines tours directly by connecting or disconnecting cities. In the semi-automatic process, the user selects a promising group of cities

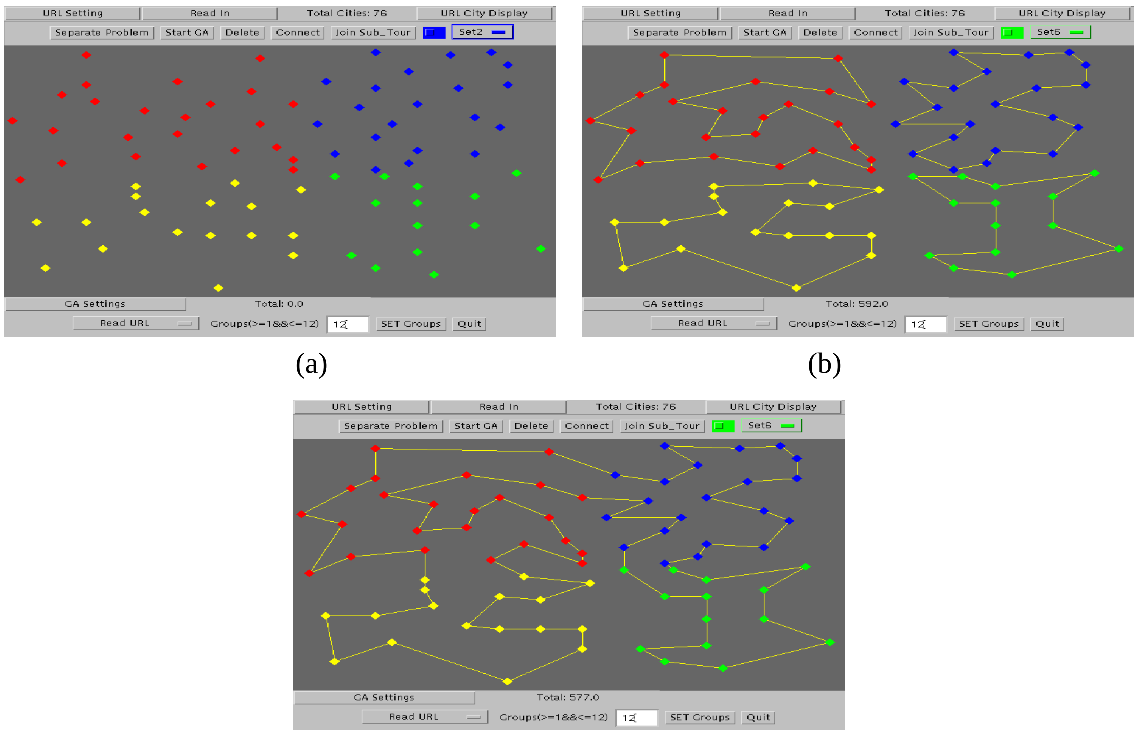


Figure 2.12: The Interactive Genetic Algorithm (IGA) approach [127]: (a) the user divides the cities into clusters; (b) sub-tours are computed for all clusters independently by using a genetic algorithm, and (c) the user combines the sub-tours to produce a global TSP solution.

belonging to two adjacent tours; the system then runs an exhaustive search on the chosen cities for finding the best way of reconnecting these elements, so that the length of the combined tour is minimized. An example of combined tour is presented in Figure 2.12(c). The IGA approach was compared against the same genetic algorithm running automatically on the whole set of cities. Experiments with benchmark data showed that the IGA could produce better tours for medium and large problem instances, and that it needed much less processing time to find good solutions for the largest problems. Louis and Tang suggest that their approach can be applied to other visually decomposable problems, and can use other types of optimization methods.

Pu and Lalanne [122, 150, 151, 152] investigate the use of interactive algorithm visualizations for solving design problems, mostly for processes that can be represented by constraint satisfaction problems. They developed a number of visualizations that provide the user with an impression of the complexity of the space of solutions. The visualizations also help to identify the search strategy employed by different constraint satisfaction algorithms. An example visualization presented by Pu and Lalanne is the *Kaleidoscope* view, illustrated in Figure 2.13. This visualization shows

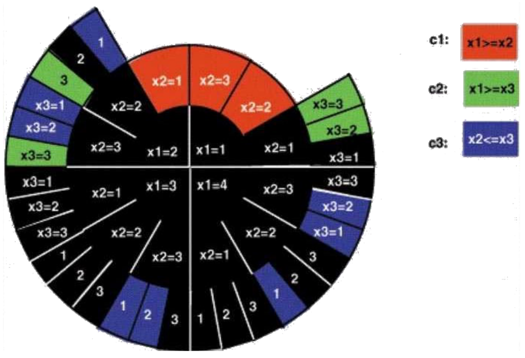


Figure 2.13: The Kaleidoscope Visualization [122].

the internal state of a search algorithm by using color patterns. The whole space of solutions is displayed as a circle, composed of concentric rings. Each ring is associated with a variable. A sector of a ring represents an assignment of a value to a variable. If the assignment does not violate any constraint, then the sector is painted in black; otherwise, it is shown with a specific color related to the violated constraint (in Figure 2.13 there are three variables, $x1$ to $x3$, and three constraints, $c1$, $c2$ and $c3$; each constraint has a different color). The circle is gradually constructed, by having a search algorithm defining assignments for the values and evaluating the satisfaction of the constraints. If an assignment violates a constraint, then the search automatically discards this path and tries another solution. This is shown by an incomplete section that only has some inner rings painted. A continuous sequence of black sectors from the inner ring to the outer ring represents a feasible solution. Several aspects of the problem can be discovered with the Kaleidoscope visualization. The user can see, for example, whether there are many or few feasible solutions, whether the search concentrates on a region of the space or is well spread, and whether there is a constraint that is more difficult to satisfy than the others. Different search algorithms are expected to present different visual patterns. Based on visualizations such as this, the user can choose the best algorithm for a particular situation, or change the problem dynamically (for example, removing or relaxing a constraint), so that a good design solution is found.

The list of domain specific interactive approaches described above is not exhaustive. Other examples include timetabling using constraint logic programming [83, 82, 139], simulation of physical processes [190], and model checking in specific logics [21].

Katchabaw *et al.* [104, 119] present an approach that is not closely related, but needs to be mentioned here. The aim of the approach is not to solve an optimization problem in an explicit and intensive way, but rather managing the allocation and usage of computational resources such as CPU power, memory, and disk and network bandwidths. The interesting point of the approach is that it uses the term “user hints” for describing the users’s interests and activities in a multi-tasking environment. These user hints can be a variety of interactive actions such as minimizing, restoring, covering and uncovering windows. An architecture is presented for collecting user hints and using them to reallocate computational resources. For instance, if the user covers a window with another program window, then the system may detect this operation and automatically reduce the number of CPU cycles allocated to the former application (if it is not a critical process). As a consequence, the window that has the immediate focus of attention of the user receives a higher priority.

2.4 Summary of Human-Computer Collaboration

All systems and general approaches described in the previous sections implement concepts of human-computer collaboration. They divide a problem into subtasks and responsibilities that are assigned to humans and to computers. The success of these approaches hinges exactly on the right choice of which tasks and roles are more suitable for humans, and which ones are better performed by automatic processes.

The tasks usually assigned to computers are:

- computing an initial solution of good quality to a problem. This is useful for well formulated problems that involve many variables to be set and/or a large solution space.
- Generating several alternative solutions, in cases where there are a number of equivalent possibilities and the system is not capable of effectively distinguishing between them.
- Improving an existing approximated solution, or completing partial solutions created by the user.
- Adjusting an existing solution in order to satisfy new constraints added by the user.
- Restricting the user’s actions so that some important constraints already implemented in the system are not violated.
- Evaluating the quality of user-generated solutions numerically.
- Providing (visual, haptic, etc.) feedback to the user. Feedback can be information about the quality of the current solution, the state of the optimization process and evaluation of the operations performed by the user.

On the other hand, humans are commonly allocated to:

- Dynamically adding, removing and adjusting constraints and objectives related to the domain of the problem.
- Evaluating the results produced by the system; for example, giving scores to computer-generated solutions.
- Providing examples of good solutions, so that the system can learn from these samples.
- Creating an initial approximate solution by hand, or a partial solution that can be improved by the system to produce a better one.
- Manually improving a solution. This is useful in cases where the improvement is intuitive to the human, but cannot be done effectively by an automatic process.
- Tuning, activating and stopping an optimization method when necessary.
- Identifying critical parts of the problem to be solved, and focusing an optimization method on these parts. This is a way of reducing the solution space to be explored by the method.

In fact, the need for having humans in the optimization process is due to a “gap” between what current automatic methods can achieve and what we would like them to achieve. In the Interactive Evolutionary systems presented in Section 2.3.1, for example, the gap is in the objective function, which is not clearly defined *a priori*. The task performed by the user aims to clarify this function by scoring proposed solutions. In the vehicle-routing system using the HuGS paradigm, in contrast, the objective function is well defined, but the optimization method merely seeks local optima and does not find sufficiently good solutions. The user’s role in this case is to guide the method towards a global minimum, through a better control of the optimization process. We can generalize these “gaps” by saying that human-computer interaction in combinatorial optimization problems has two major goals:

- *Refining the optimization problem* – that is, inserting domain knowledge into the system so that it better represents the user’s interests or the real-world problem to be solved. This can be done by adjusting the set of constraints and the objectives of the problem. Such adjustment can be done directly (as in the Constrained Graph Drawing Using Springs presented in Section 2.2, where the user manually specifies layout constraints) or indirectly (as in the

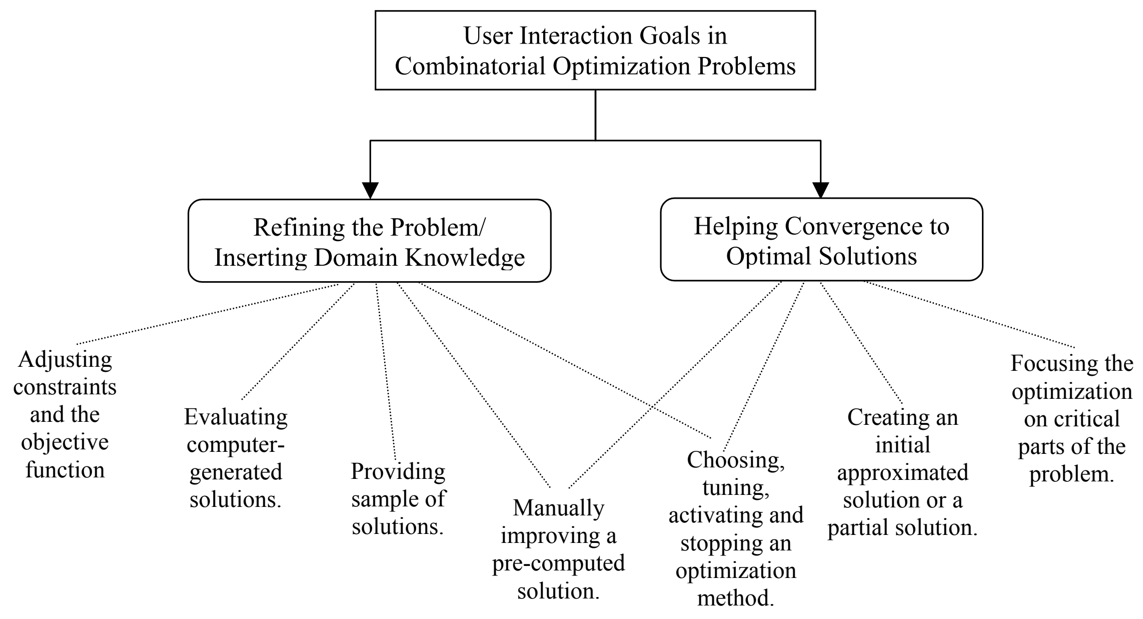

User Interaction Goals in Combinatorial Optimization Problems
Refining the Problem/ Inserting Domain Knowledge
Helping Convergence to Optimal Solutions
Adjusting constraints and the objective function
Evaluating computer-generated solutions.
Providing sample of solutions.
Manually improving a pre-computed solution.
Choosing, tuning, activating and stopping an optimization method.
Creating an initial approximated solution or a partial solution.
Focusing the optimization on critical parts of the problem.

Figure 2.14: User actions classified according to the two major goals.

Interactive Evolutionary systems in Section 2.3.1, where the user provides scores or samples of solutions, and the system uses this information to tune an objective function).

- *Helping convergence to optimal solutions* – helping the system to obtain better solutions to a given problem, and/or produce solutions of a high quality in much faster ways. This is the goal of the Haptic Hints approach, the HuGS paradigm, and most of the other systems presented in Section 2.3. Such a goal considers a fixed objective function and a fixed set of constraints.

In Figure 2.14, we present the relationship between the user actions and these two goals, as it has been defined by the approaches described in this chapter.

Note that some interactive actions are helpful for both goals. Moreover, the figure does not show all possible links: we know that some user actions linked only to one goal may also be useful to the other goal in particular applications. For example, temporarily relaxing constraints can be helpful to improve convergence to better solutions.

Another important element in interactive systems is the existence of visualization tools, which must provide interactive facilities and visual feedback for the users. All systems studied in this section make use of some sort of visualization tool.

CHAPTER 3

The User Hints Framework

This chapter presents our interactive *User Hints framework* for Combinatorial Optimization Problems. The framework aims to achieve the two major interactive goals described in Section 2.4, refining an optimization problem and improving convergence. This is achieved via a combination of features from Human-Computer Interaction, Combinatorial Optimization, and Information Visualization.

3.1 The Elements of the Framework

Figure 3.1 shows a diagram of the User Hints framework. The framework involves nine elements: (1) a user who is a domain expert, (2) a set of objectives (that is, objective functions) and (3) a set of constraints that compose a combinatorial optimization problem, (4) a module with optimization methods, (5) a solution that is currently being improved, called the *working solution* (this is simply an assignment of values to the variables of the problem), (6) a quality function (not represented in the picture) that measures the quality of a solution, (7) a *best solution agent* that saves internally the best working solution computed so far, (8) a visualization tool, and (9) a visualization created by the visualization tool that provides feedback about the working solution and the state of the optimization[1]. The connection between the elements of the framework is represented by arrows. Dashed arrows indicate dependencies that we often consider implicitly in customized versions of this diagram in the next chapters.

An optimization process using the User Hints framework consists of setting the objectives and having the user interact with the elements of the framework in order to produce a solution of good quality. More precisely, the optimization process works as follows:

[1]Note that we use the term visualization to cover all kinds of perceptual feedback, including nonvisual kinds such as sound and haptics.

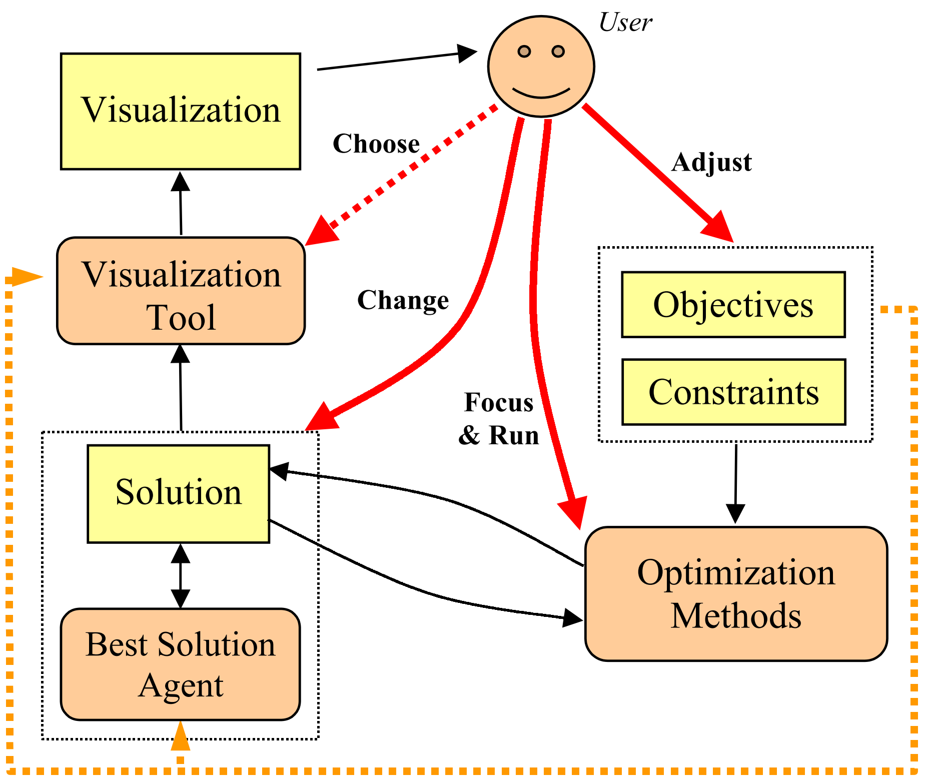


Figure 3.1: The User Hints framework.

(a) An initial working solution is automatically created, and the visualization tool is initialized to provide a visualization of this solution. The best solution agent is also activated, and it saves a copy of the initial solution as the best solution.

(b) The system then waits for an action of the user, who is the controller of the optimization. The user can either interact with the visualization tool to choose a different visualization, or give *hints* to the optimization process. Hints are adjustments aiming to refine the problem by inserting domain knowledge, to reduce the space of solutions to be explored, to escape from local minima, or to immediately produce a better solution. Such hints may change the working solution.

Any change of either the working solution, the constraints or the objectives automatically triggers the visualization tool and the best solution agent. When triggered, the visualization tool redraws the picture of the working solution. The best solution agent, on the other hand, compares the quality of the working solution with the quality of the best solution (saved internally), and updates the best solution if necessary.

The quality of a solution is determined by the quality function, which considers objectives and constraints of the problem.

Moreover, whenever the best solution is updated, the best solution agent triggers the visualization tool to provide an immediate feedback to the user about this event.

(c) At the end of the optimization process, the best solution is considered the final solution of the problem.

This process repeats until the user is happy with the result.

3.1.1 Types of Hints

We consider three main types of user hints:

- *Adjustment of objectives and constraints* – users can change the objective function as well as add new constraints or remove existing ones. This is useful for inserting domain knowledge into the problem after starting the optimization process.
- *Focus of an optimization method* – the idea is to focus the action of the optimization method on particular areas of the problem that need major improvement. Focus is implemented by allowing the users to select a subset of variables of the problem, and then running the optimization method to change only this subset; the remaining variables are kept fixed. Since the complexity of solving a sub-problem is in general smaller than that of the whole problem, using this type of hint helps to reduce the solution space.
- *Manual changes* – all optimization aspects that are not covered by constraints and focus can be managed by manual changes. This is useful, for example, when the user sees simple modifications of the solution that can lead to a significant improvement in quality. Manual change is allowed by directly altering the working solution.

Beside these types of hints, the user can also perform three other actions:

- *Recovering the best solution* – the user calls the best solution agent to replace the working solution with the best solution.
- *Setting the best solution* – the user calls the best solution agent to save the working solution as the best solution. The current best solution is replaced even if it has a higher quality than the working solution.

- *Controlling the optimization module* – the user can control the optimization module by choosing a different optimization method to be used, directly stopping the execution of an optimization method that was left running, or adjusting some tuning parameters of the method.

The last three interactive actions are considered indirect types of hints, since they only effect the optimization process when combined with the main types of hints.

Next we describe details of some of the elements of the User Hints framework.

3.1.2 Constraints and Feasibility

The framework can start with some basic constraints from the problem domain, and allow the user to adjust the constraint set on the fly. The optimization methods and the user work to create a solution that satisfies the constraints.

Two approaches exist for controlling the satisfaction of a constraint:

- *Allowing infeasible solutions to be generated, which temporarily violates the constraint.* This is useful, for example, when it is more natural to improve a solution by temporarily violating some constraints than by working only on the feasible solution space. The user may create an infeasible solution by performing manual changes of a feasible solution, or by adjusting the constraint set. In the latter case, a new constraint added to the constraint set is violated by the current working solution or by the best solution.

- *Always satisfying the constraint.* In this case, user actions such as manual changes and constraint adjustments are not permitted if they imply infeasible solutions. Another possibility is to allow such user operations, but immediately and automatically post-processing the working and the best solution to guarantee feasibility.

The best choice of the possibilities above depends on the optimization problem, the nature of the constraint, and on the type of human interactions that is intended in the framework.

3.1.3 The Initial Solution

The initial solution can be created in one of the following ways:

- Using a simple algorithm to produce a trivial solution or a random solution, possibly an infeasible one.

- Using a good constructive method from the literature that provides an initial solution of high quality.
- Creating a trivial or random solution, and automatically improving it by calling an optimization method implemented in the framework.
- Recovering a pre-computed solution that has been saved.

3.1.4 Quality Function

The quality function computes the quality of a solution based mainly on the objectives of the problem. If infeasible solutions are allowed, then the quality function may also measure constraint violation by assigning higher quality values to feasible solutions than to infeasible ones.

3.1.5 Optimization Methods

Any type of optimization method can be used in the framework. These include:

- Non-improvement methods – start with the constraints and the objectives of the problem, and produce a solution. Examples are constructive methods such as some problem-specific heuristics and greedy heuristics, and Integer Linear Programming (ILP) techniques.
- Improvement methods – input an initial solution, and improve it according to the set of constraints and objectives. Examples are some local-improvement meta-heuristics such as Hill Climbing, Simulated Annealing, and Genetic Algorithms.

Ideally the optimization methods should compute solutions that optimize the quality function of the framework. However, methods that implement simpler objective functions or consider a smaller subset of the constraints may also be applied, when no better algorithm is available.

All optimization methods in the framework must be adapted to support focus. This means that they will only change the values of variables that were selected by the user.

Moreover, the methods should allow a cyclic (iterative) improvement of the working solution. Such a cycle is naturally obtained when using improvement methods. It can also occur by combining any method with the focus mechanism in order to change only a particular area of the solution. Finally, the cycle may be formed by using the quality of the current working solution as a bound to the quality of every new solution. This last option can be useful, for example, in branch-and-bound algorithms for solving ILP problems.

3.1.6 The Visualization Tool

The Visualization tool has two functions: providing feedback of the optimization process to the user, and allowing the user to directly change the elements of the framework.

The first function is performed by offering:

- A picture of the working solution – the picture must emphasize the quality of this solution by using different colors or shapes; this can present constraint violation and objectives that are poorly achieved.
- Numerical information about the quality of the best solution and of the working solution, for comparison.
- Indication that the optimization method is currently running. This can be done by showing a progress bar, or by replacing the working solution with samples of the intermediate solutions being produced by the method.
- Indication that the best solution was updated with a new working solution of better quality.

The second function of the visualization tool is concerned with supporting direct manipulation. Instead of having the user changing the working solution or adjusting constraints via a complex textual interface, the visualization tool must support direct manipulation of the visualization for this aim[2]. Selection of variables for focusing, for instance, may be implemented by allowing the user to click on graphical objects that represent these variables. Immediate feedback is necessary to indicate that the user action was performed. For the focusing case, the visualization may respond to user clicks by changing the color of the objects that were selected.

The user must also be able to interact with the visualization tool in order to change the visualization (in case more than one visualization is available) or to adjust it, such as zooming in or out, and hiding or showing elements of the picture.

Note that the visualization tool is a critical element of the framework, since the user identifies quality aspects of the solution, and decides what action to perform next, based on the feedback that he or she receives from this tool.

[2]This implies that the user interaction with the elements of the framework, represented by arrows leaving the user in Figure 3.1, occurs in fact through the visualization tool.

3.1.7 Work Modes

The framework supports human-computer task division in a number of different work modes that match the interaction paradigms presented in Section 2.2.1:

- *Fully manual optimization* – the user creates a solution for the optimization problem by performing only manual changes. The optimization methods are not executed.
- *Fully automatic optimization* – An optimization method is automatically activated by the framework to improve the initial solution over a period of time. All variables of the problem are selected, so that the method works on the entire solution. Optionally, the user can be responsible for activating the method, but he or she does not perform any other task.
- *Manual post-processing* – this is a step further from the fully automatic approach, where the user improves the automatically-generated solution by performing manual changes on it.
- *Improvement-based optimization* – this approach is the opposite of a manual-postprocessing. Here, the user firstly produces a solution by hand, and then calls an optimization method to improve the whole solution.
- *Constraint-based optimization* – the user changes the constraint set, and calls the optimization method to adjust the working solution in order to satisfy new constraints. A different approach is to allow the users to perform only manual changes, but restricting their actions so that a predefined set of constraints is not violated.
- *Collaborative optimization* – This integrates the possibilities described above plus all other resources of the framework (such as the focus mechanism). The user can change a solution manually, adjust the constraint set and the objectives, and choose, focus, execute and stop an optimization method. The collaborative mode is characterized by having the user performing direct changes of the solution or the problem together with executions of the optimization methods.

These work modes are not defined explicitly in the framework; rather, they are general paradigms that can be adopted intuitively by the user. We are, however, more interested in motivating the user to adopt the collaborative mode, where a combination of human skills with computational processing may provide better results.

An interesting point to note here is that the framework always provides some help to the user, independently of how he or she approaches the optimization. This help comes mainly in the form of a continuous visualization of the state of the optimization process. Moreover, the best solution agent helps the user by filtering, signalizing and saving the best solution generated so far, so that the user can return to this solution if necessary.

Another point is that some elements of the framework have a high degree of autonomy and can execute tasks in parallel with the user. These elements are shown by rounded boxes in Figure 3.1. For instance, it may be possible to have an optimization method improving a part of the working solution, while the user is manually changing another part.

3.2 Comparison with Other Approaches

In this section we compare other approaches for interactive optimization presented in Chapter 2 to the User Hints framework.

Interactive Evolutionary Systems [180], as introduced in Section 2.3.1, share with the User Hints framework the goal of refining the optimization problem. However, such systems define quite a passive role for the user, who must provide inputs to the optimization method when requested. The user does not have direct control of the constraints and objectives of the problem; rather, he or she indirectly adjusts these elements by acting as a fitness function or by providing handmade sample solutions. Moreover, little support exists for helping convergence; direct control of the execution of the optimization methods is in general not considered important.

In *Haptic Hints* [13, 14], the user's task is to help convergence. The user plays an active role by creating an approximate or partial handmade solution, which is directly used by the system to produce the final solution to the problem. These features are also included in the User Hints framework[3]. The difference is that the Haptic Hints approach has been proposed for motion planning problems, while our framework is for optimization problems in general. Haptic Hints also does not offer a general focus mechanism; focus on a particular region of the problem is done indirectly by providing approximate and partial paths. Furthermore, Haptic Hints puts emphasis on having

[3]Note that the concept of a partial solution can be modeled in our framework. One possibility is to define a solution as consisting of two sets: a set of variables of the problem that were already assigned a value, and a set of variables that have not yet been initialized. Constructive optimization methods could be used to move variables from the second to the first set by assigning values to them. In this case, a precondition for feasibility of a solution could be to have all variables assigned a value – that is, the second set would be empty. Another approach is to define a special value, so that variables assigned to such value are considered not initialized. This second configuration is similar to the approach that we use in the Chapter 7 for selective labeling of point features in maps.

the users interacting at the begin of the optimization process. The user does not interfere with the improvement process of the handmade solutions and cannot add constraints during processing time.

Human-Guided Search (HuGS) [8] is quite close to our framework. It implements the same basic principles of focus and control of the optimization method as in the User Hints framework. However, there are some differences.

One difference is that HuGS offers a more refined version of the focus mechanism, based on three levels (low, medium and high) of mobility for elements of the problem. On the other hand, we establish only two conditions for focusing on variables, *selected* and *unselected* conditions, which are equivalent in many cases to high and low mobilities, respectively.

HuGS also emphasizes the ability to return to previously computed solutions, saved in a history list, as one of their main interactive resources. We implement a similar feature, but it is proposed as an active agent that identifies and saves the best solution, allowing the user to recover or replace it afterwards. The idea of an active agent that oversees all solutions produced by the user or the optimization methods, and promptly feedbacks the user when there is an improvement of the best solution is conceptually different from a passive history. A point to note is that, at the moment, our agent saves only one solution; an extension for the agent is proposed in Section 8.5 of this thesis.

The main difference between the HuGS and the User Hints framework, however, is that HuGS aims to help convergence, while our framework is for both convergence and problem refinement tasks. Also, early versions of HuGS used a simple Hill Climbing heuristic, and was extended later to include tabu search. We have not restricted, in contrast, the type of algorithm that can be used. In fact, we present interactive systems that implement problem-dependent heuristics, Hill Climbing methods, Simulated Annealing, and Genetic Algorithms.

Despite the differences, all interactive approaches described above are relatively new, and are currently evolving to incorporate new features. Undoubtedly they will share many successful ideas in the future. The study of Interactive Optimization is only in its formative stage; the main contribution of our research is to provide more experience to help consolidate the area.

Nomenclature

Note that the term “hints” has been used by a number of authors (in Haptic Hints [13, 14], in the Maze router presented by Arnold and Scott [9], and in a resource management approach for multitasking operating systems [104, 119]). These hints have a common characteristic: they are inputs that control the optimization process indirectly. In some cases, the hints are information obtained by the system in observing the user and trying to identify his or her intentions. In other situations, hints are explicitly provided by the user, but they roughly approximate or give incomplete clues about final solution to be produced.

In contrast to this kind of indirect action, the users of the systems described in this thesis directly control the optimization process.

In the following chapters, we present approaches for specific optimization problems based on the User Hints framework.

CHAPTER 4

User Hints for Graph Clustering

This chapter presents our first investigation of the User Hints framework. We apply the framework for the Graph Clustering Problem, which is a NP-hard optimization problem that arises in many applications. An interactive Graph Clustering prototype system based on user hints is described.

For being a preliminary study, the Graph Clustering case is quite simple and does not explore all possibilities of the interactive framework. The prototype also lacks some major resources that would improve its efficiency and effectiveness – it may not produce good clustering solutions even for some simple graphs. Nevertheless, the study allowed us to identify promising interactive facilities and paved the way for future applications of the User Hints framework. The main lessons we learned are discussed here.

This chapter is organized as follows: Section 4.1 introduces the Graph Clustering problem; Section 4.2 presents the interactive framework for Graph Clustering based on user hints; Section 4.3 describes a prototype system developed for experimenting with the framework; finally, Section 4.4 discusses the use of the system and the lessons we learned with it.

4.1 Graph Clustering

Graph clustering, also known as Graph Partitioning, aims to divide the set of vertices of a graph into disjoint subsets (clusters or partitions) while minimizing the connection between vertices in distinct sets and satisfying some constraints. Formally, a clustering S of a graph $G = (N, E)$ (where N is a set of weighted vertices and E is a set of weighted edges) is a partition of N into disjoint subsets (called clusters) N_1 , N_2,... N_k. There are a number of clustering problems, each with the following form: find a clustering S of G, subject to some constraints, such that a measure of the intercluster edges (that is, edges (u, v) with $u \in N_i$, $v \in N_j$, $i \neq j$) is minimized. The variants of the problem come from choosing k, the constraints, and the measure of the intercluster edges. For example, a common variant is to fix the number k of clusters and impose a “balance” on the size of the clusters. This problem is called k-*way partitioning* [69]. When $k = 2$, we have a special case known as *bisection* or *bipartitioning*. Several other kind of constraints can be specified for the problem, such as to limit the maximum weight of the vertices in each cluster.

Almost all variations of the graph clustering problem are NP-Hard (even the bisection case [73]); therefore, heuristics are the most used methods for providing clustering solutions.

Graph Clustering has several applications. It is necessary, for example, in distributed and parallel processing [120] for dividing a large set of tasks into subsets so that each one can be allocated to a different processor; here tasks are seen as vertices of a graph and the dependencies between the tasks are modeled by edges; the aim is to assign tasks to each processor in order to achieve a balanced load and to minimize the communication between pairs of processors.

Another application of clustering is in VLSI design [5]: a circuit is divided into blocks with few connections between them via clustering methods, thus reducing the complexity of dealing with the whole structure.

Finally, Graph Clustering is also important in Software Engineering, where it is used to divide huge programs into packages with high internal “cohesion” , and with loose “coupling” with other packages. In this application, Graph Clustering methods can be applied on the flow graph of the program, on a call graph, on a control flow graph, or on any one of the many graphs used in object-oriented design. Many clustering methods have been used for this purpose; see, for example, [117, 157, 186]. Graphs arising from legacy code are particularly interesting for clustering for two main reasons:

- *Program comprehension*. Human understanding of legacy software is a problem that has become critical in recent years. A legacy system often consists of thousands of interdependent functions, and the human must understand these dependency relations. Clustering reduces the amount of information to be understood, and allows the human to think in terms of higher-level architectural dependencies rather than at the function level.

- *Re-factoring*. Code that has been maintained, updated, adjusted and ported over a period of several years has a tendency to lose the elegant structure that it once had. Clustering the code units (terms, functions, or files) can suggest new structures for the code.

In general, clustering is important for all kinds of problems where either the size of the graph is big and has to be reduced, or some inherent structure of the graph can be discovered by looking for coherent parts.

There are several methods for solving Graph Clustering problems. Some of the most popular ones are:

- Meta-heuristics: these include greedy strategies (for example: Kerninghan-Lin algorithm [105], Fiduccia-Mattheyses algorithm [68], and Sanchis' algorithm [169]), Simulated Annealing [100], Tabu Search [160], and Genetic Algorithms [30, 167, 193].
- Spectral partitioning: partitioning a graph according to the eigenvalues of the incidence matrix can be very effective (a review of spectral partitioning algorithms is given in [5, 63]).
- Flow methods: identify minimum "cuts" in the graph, on which a clustering can be based.
- Integer linear programming methods: it is fairly straightforward to encode the quality requirements and the constraints of a clustering problem in a linear fashion, and then use methods from integer linear programming [64, 87].

4.2 An Interactive Framework for Graph Clustering

In the literature of Graph Clustering human interaction is not strongly integrated with automatic clustering methods. In general, human interaction happens before running the methods – in order to define a clustering problem– or after the execution – to refine the problem when it needs adjustments. In the latter case, the method is usually re-executed on the modified problem to produce a new clustering solution, and the previous solution is discarded.

In this section we present a flexible interactive framework where users incrementally refine a graph clustering problem and improve an existing clustering solution. Note that most of the clustering algorithms presented in Section 4.1 are heuristics and, therefore, there is a place for users to improve approximate solutions. At runtime, the users can visualize the current clustering and re-execute a clustering algorithm. The users perform two tasks:

1. *Insertion of domain knowledge*: the visualization of the current clustering may lead the user to believe that, although it satisfies the current constraints and has a good value for the objective function, it is not "right" in the domain context. The user must give the algorithm a hint to move toward a solution that is correct in the domain context. As an example, suppose that the vertices represent modules of a software system, and the edges represent data-sharing

relationships between the modules. The user may know that two specific modules share data in a way that is not captured by the formal graph model extracted from the source code, and may use this knowledge by forcing these two specific vertices into the same module. As another example, the user may see that the constraints on cluster size are too tight to result in a clustering with good cohesion; in this case the user can relax the constraint.

2. *Guidance of the search*: the user may see that the algorithm is spending a considerable amount of time in small adjustments of a cluster with very poor cohesion, or that it is not improving the clustering at all. In those cases, it is possible to perform many interactive actions in order to help the algorithm to improve the clustering.

The user can do these tasks by giving hints to the system in two main ways:

- *Adjusting constraints* – constraints are an essential part of clustering, as we saw in the definition of the k-way partitioning problem in the previous section. However, for strong user involvement we need to allow not only the definition of constraints prior the clustering process, but also allow dynamic adjustment of constraints at run-time (adding new constraints, removing old constraints, or changing the importance of a constraint). By changing the constraints at run-time, the user can guide the algorithm to converge to a different solution. Simple examples of global constraints that can be specified are: minimum and maximum numbers of clusters, bounds on the number of vertices in each cluster, and limits on the variation of the clusters size. There can also be local constraints, for example: having two particular vertices a and b always in the same cluster, or in different clusters.

- *Direct manipulation* – the user can directly operate on the clustering by destroying an existing cluster or merging two clusters. Destroying a bad cluster and forcing the algorithm to reassign its vertices to other clusters is a way of moving out of a local minimum.

Another type of hint that is not so intuitive as the previous ones, but that can be helpful is to choose a different clustering method. The user selects a more appropriate clustering algorithm whenever the current method cannot improve the quality of the actual solution. In this case, the user may even decide to run the algorithm on the whole clustering or focus it just on part of the clustering solution that shows poor quality. Several algorithms may be available such as the ones presented in Section 4.1. Algorithms that work with a wide range of constraints and can support continuous improvement of a solution are, however, more suitable for the job. Meta-heuristic methods are promising candidates.

Our User Hints framework for Graph Clustering follows the diagram in Figure 3.1, in the previous chapter. We assume that the user supplies hints through a graphical interface. The interface has to be implemented in a way that it facilitates user interaction and provides a meaningful visualization to the user.

The framework consists of six elements: a graph G to be clustered, an objective function O, a set C of constraints, a clustering S of G, a vector Q that measures the quality of S according to O and C, and a visualization P of S. The dependence between these elements as well as the order in which they are set are shown in Figure 4.1. The framework involves three steps (also shown in the figure):

1. *Setup or adjustment step* – the user defines the graph G and specifies O and C. These elements can be adjusted later if necessary.

2. *Clustering step* – a clustering S is produced (or improved, if it already exists) by invoking a graph clustering algorithm. The algorithm aims to minimize O and to satisfy C. After the processing, the quality Q of the clustering is computed.

3. *Visualization step* – a visualization P of the clustering is then automatically created. The visualization also uses the attributes of the vector Q to highlight to what extent S satisfies the constraints and the objective function.

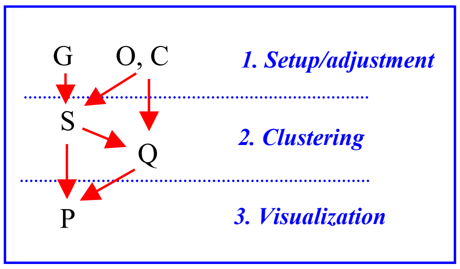


Figure 4.1: The elements and the steps involved in the User Hints framework for Graph Clustering.

These steps are executed sequentially at the first time, so that all elements of the framework are created. The user can then re-execute step 1 or 2, or review the whole processing in order to improve the clustering. Note that this may be necessary if the initial clustering is not the global optimum for the problem, or if it is not feasible (that is, it does not satisfy all constraints in C).

It may also be the case that the user has specified the constraint set improperly and, therefore, the clustering solution is not what he or she was expecting.

Human interaction, aiming to improve the clustering, can be done by adjusting C or directly changing S through the interface. Any modification of these elements forces Q and P to be recomputed to reflect the changes.

Note that some steps in the framework can be executed in parallel, as suggested in section 3.1.7. For example, the user can adjust the constraint set while the clustering algorithm is running; this should alter the clustering computation instantly.

4.3 The ClusterHints System

In order to experiment with the framework we developed a system called ClusterHints (Figure 4.2 shows a snapshot). The system implements some of the features described in the previous section. Its main resources are:

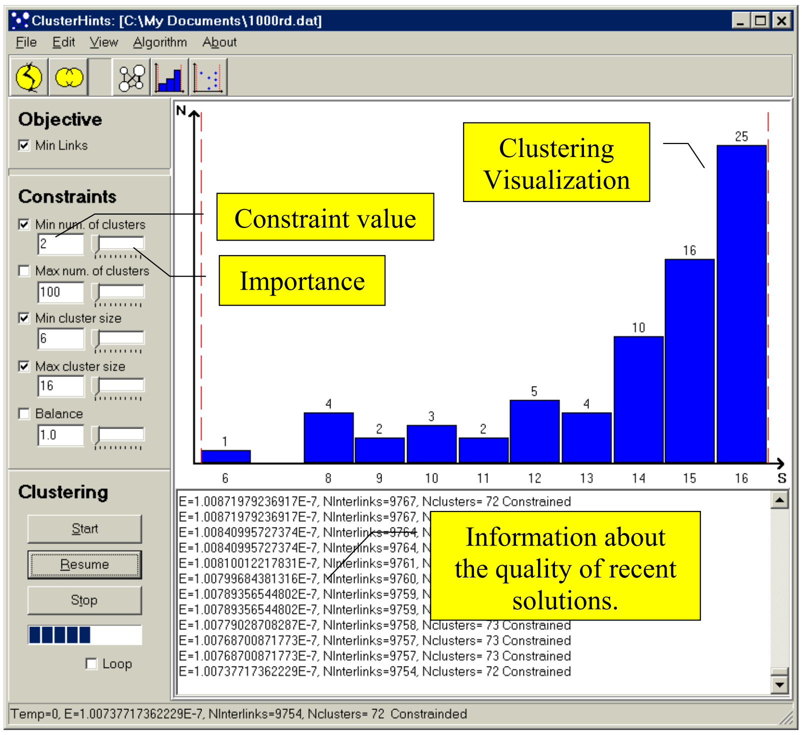
- a graphical user interface for setting global constraints;
- two clustering algorithms, a Simulated Annealing and a stochastic Hill Climbing method, that allow dynamic changes of the constraint set;
- a visualization tool that shows drawings and measures of the clustering solution.

The focus facility and the best solution agent were not implemented.

Next we explain details of the ClusterHints system.

Constraints

The system supports a set $C = (C_1, C_2, \ldots, C_5)$ of five constraints. Each C_i, $1 \leq i \leq 5$, consists of a pair (l_i, w_i) of numbers, where l_i is the value of the constraint, and w_i represents its importance. The constraint importance is a real number between 0 and 1, with 1 representing the highest importance, and 0 meaning that the constraint is not considered in the optimization. The values l_i, $i = 1, 2, \ldots, 5$, depend on the meaning of the constraints, which are described as follows: C_1 and C_2 define the minimum and the maximum numbers of clusters, respectively, to be produced during the clustering processing; C_3 and C_4 are limits on the minimum and maximum sizes of every cluster (they impose an upper and lower bound on the number of vertices inside a cluster); and C_5 forces



☑ Min Links

☑ Min num. of clusters: 2
☐ Max num. of clusters: 100
☑ Min cluster size: 6
☑ Max cluster size: 16
☐ Balance: 1.0
☐ Loop

```
E=1.00871979236917E-7, NInterlinks=9767, Nclusters= 72 Constrained
E=1.00871979236917E-7, NInterlinks=9767, N
E=1.00840995727374E-7, NInterlinks=9764, N
E=1.00840995727374E-7, NInterlinks=9764, N
E=1.00810012217831E-7, NInterlinks=9761, N
E=1.00799684381316E-7, NInterlinks=9760, N
E=1.00789356544802E-7, NInterlinks=9759, N
E=1.00789356544802E-7, NInterlinks=9759, N
E=1.00779028708287E-7, NInterlinks=9758, Nclusters= 73 Constrained
E=1.00768700871773E-7, NInterlinks=9757, Nclusters= 73 Constrained
E=1.00768700871773E-7, NInterlinks=9757, Nclusters= 73 Constrained
E=1.00737717362229E-7, NInterlinks=9754, Nclusters= 72 Constrained
```

```
Temp=0, E=1.00737717362229E-7, NInterlinks=9754, Nclusters= 72  Constrainded
```

Figure 4.2: The ClusterHints system for Graph Clustering.

a balance of the clusters' sizes, by defining the size of the largest cluster divided by the size of the smallest cluster to be smaller than a given factor (specified by l_5). The user can change the value and the importance of the constraints through a panel in the interface.

Initial Clustering

The system automatically creates an initial clustering S for a graph G specified by the user. The initial clustering is a trivial solution that consists of one single cluster containing all vertices of the graph. The quality and a visualization of the clustering are also automatically computed.

Quality of the Clustering

The quality of a clustering S is a vector $Q(S) = (L(S), f_1(S), f_2(S), f_3(S), f_4(S), f_5(S))$, where $L(S)$ represents the objective function and gives the number of edges between vertices in distinct clusters, and $f_i(S)$, $1 \leq i \leq 5$, are functions that measure the satisfaction of the constraints $C_i = (l_i, w_i) \in C$. Let k be the number of clusters in S, $S = (N_1, N_2, \ldots, N_k)$, then the functions f_i are defined as:

- $f_1(S) = max(0, l_1 - k)$

- $f_2(S) = max(0, k - l_2)$
- $f_3(S) = \sum_{j=1,2,...,k} max(0, l_3 - |N_j|)$
- $f_4(S) = \sum_{j=1,2,...,k} max(0, |N_j| - l_4)$
- $f_5(S) = max(0, \frac{max_{j=1,2,...,k}(|N_j|)}{min_{j=1,2,...,k}(|N_j|)} - l_5) * (1 + \frac{X+Y}{|V|})$, where X and Y are the numbers of clusters with smallest and greatest size, respectively.

Instead of using Q directly to measure the quality of a clustering S, we compute a weighted function $\epsilon(S)$ involving the constraints and the objective function:

$$\epsilon(S) = b * O(S) + \sum_{1 \leq i \leq 5} \frac{w_i * f_i(S)}{\lambda_i},$$

where b is a constant, sufficiently small for making the constraints always more important than the objective function $O(S)$. The objective function in this case computes the number of intercluster edges. The parameters λ_i, for $i = 1, 2, \ldots, 5$, are the largest values that can be defined by $f_i(S)$. Small values of $\epsilon(S)$ are associated with high quality clusterings.

Executing a Clustering Algorithm

A panel of the graphical interface allows the user to start, pause, resume, and terminate a clustering algorithm. The algorithm can be the Simulated Annealing and the Hill Climbing method. Both algorithms work iteratively on the elements of the framework specified in the previous section. They basically input G, O, C, S and Q, try several improvement steps, and update S and Q if a better clustering solution is found. The processing repeats for a fixed number of iterations, based on the size of the graph. A progress bar indicates how much processing is left.

The code for a single iteration of the stochastic Hill Climbing algorithm is presented below:

```
1. $S' \leftarrow S$;
2. $n \leftarrow k$;
3. Repeat $n$ times:
    a) Change $S'$ randomly;
    b) If $\epsilon(S') < \epsilon(S)$ then $S \leftarrow S'$; else if $\epsilon(S') > \epsilon(S)$ then undo the previous change of $S'$.
```

The parameter k is the number of clusters in S before executing the loop at line 2. A change of S is made by transferring a group of vertices from one cluster to another randomly. The transfer may also create a new cluster or delete an existing one.

The Simulated Annealing is similar to the Hill Climbing, except that it accepts worse solutions during its internal loop. Nevertheless, the clustering S is only updated if a better solution is computed. The algorithm uses a variable T, representing an annealing temperature that decreases over time by a rate r. The initial value of T and the rate r are set when the user activates the Simulated Annealing. Each iteration of the algorithm works as follows:

1. $S' \leftarrow S$;
2. $n \leftarrow k$;
3. Repeat n times:
    a) $z \leftarrow \epsilon(S')$;
    b) Change S' randomly;
    c) If $\epsilon(S') < \epsilon(S)$ then $S \leftarrow S'$; else if $random(0..1) > exp(-(\epsilon(S') - z)/T)$ then undo the previous change of S'.
4. $T \leftarrow T/r$;

Giving Hints

The user can use the control panel on the left-hand side of the screen in order to adjust the values l_i and w_i for every constraint $C_i \in C$. Note that this is done in a very intuitive way, so that the user does not have to worry about the details of the function ϵ being optimized. It is also possible to merge two or more clusters, or to break a large cluster down into single vertices. If the user makes any change when the clustering algorithm is running, then the next iteration of the algorithm will work on the newly modified data.

Visualizations

The interface offers feedback to the user through three main visualizations:

- a histogram chart that shows how many clusters exist in S for some size values;

- a scatter-plot graphic showing the size of each cluster in S; and

- a drawing of the graph implied by the clustering – the system builds a *clustering graph* $G_S = (V_S, E_S)$, where V_S is a set representing the clusters in S, and E_S is a set of weighted edges between pairs of clusters. The clustering graph is constructed as follows: every cluster $N \in S$ is assigned to a unique vertex $V_N \in V_S$; for every distinct pair of clusters N_i and N_j in S, if N_i and N_j contain vertices of the original graph G that are connected by an edge, then an edge (V_{N_i}, V_{N_j}) is inserted into E_S. The weight of every edge $(u, v) \in E_S$ is given by the number of intercluster edges connecting vertices of the original graph assigned to the clusters represented by u and v. A modified version of the spring algorithm [57] is used to produce a drawing of the clustering graph. Clusters are drawn as circles; the size of each circle is proportional to the number of vertices in the cluster. Each circle is assigned an integer which uniquely identifies the cluster. The X, Y-positions of the center of the circles are defined so that pairs of clusters connected by an edge are drawn closer to each other on the picture, and their distance is based on the weight of the edges.

These visualizations are illustrated in Figure 4.3. They also highlight some aspects of constraint satisfaction by using colors or drawing other elements on the picture. For example, the histogram chart and the scatter-plot graphic show dotted lines (see Figures 4.3(a) and 4.3(b)) representing the minimum and maximum limits on the size of the clusters. The graph drawing picture (Figure 4.3(c)) paints the borders of clusters whose sizes are out of the desired limits in red color. These simple graphical attributes call the user's attention to problems with the current clustering solution.

The user can swap between these three views in order to to visualize different aspects of the clustering.

The graphical interface also includes two other types of visualizations, which are quite simple:

- A textual area with the history of the main clustering solutions produced by interacting with the system (manually merging or breaking cluster, or changing the constraints) or by running a clustering algorithm. The history shows the value of ϵ, the number of intercluster edges, the number of clusters and whether the clustering satisfies all constraints or not.

- A status bar at the bottom of the interface displays a brief description of the quality of the clustering being improved by a clustering algorithm (this is important particularly for the Simulated Annealing, since intermediate solutions and the temperature value are not shown in the other types of visualization).

In general, the processing of the algorithms is indicated by continuously updating the visualization whenever a new good clustering solution is produced. If the screen is not updated for a long time, this may indicate that the system has reached a local minimum and requires user intervention.

4.4 Remarks

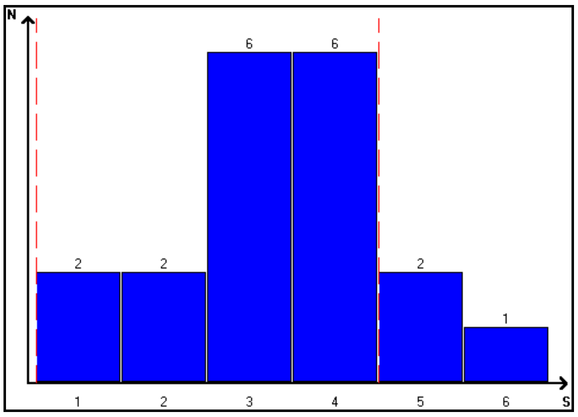
This section discusses some lessons learned by experimenting with the interactive system. We performed clustering tasks with graphs containing 40 to 1000 vertices; these included randomly-generated graphs, and graphs that had a clear structure such as grids and trees. Different combinations of constraints were tried on the size, balance and number of the clusters. The constraints were sometimes conflicting. We also experimented with the facility of manually merging and breaking clusters when the system could not improve a clustering, in order to escape from a local minimum. Interaction with the system (changes of the constraints and of the clustering) was done in two ways: after the algorithm had complete the execution, and during run time.

The interactive facilities and the visualizations implemented in the prototype showed some benefits:

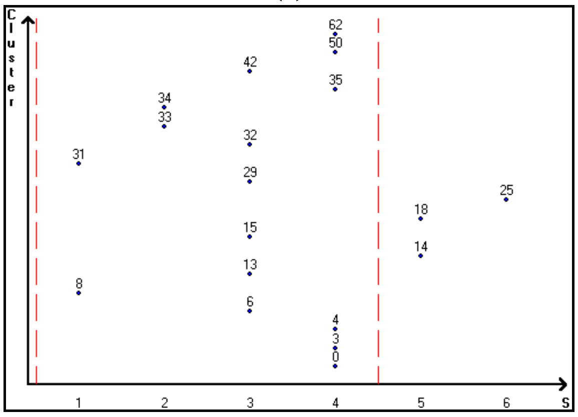
- Adjusting the constraint set was useful in cases where the system could not satisfy a particular constraint without sacrificing the remaining ones. Some successful approaches for this problem were to temporarily deactivate a subset of the constraints (by setting their w-values to zero), decreasing the w-values of the constraints that were intuitively less important, and relaxing the problem by creating easier constraints to solve (for instance, defining a wider range for the size of the clusters).

- Allowing the user to start and stop the clustering algorithms manually at any time provided more control of the optimization task. It was possible to re-execute the algorithms to improve a clustering when the specified number of iterations was not sufficient to produce the desired solution; moreover, the algorithms could be stopped when there were still many iterations to complete, but the rate of improvements being made on clustering was not significant.

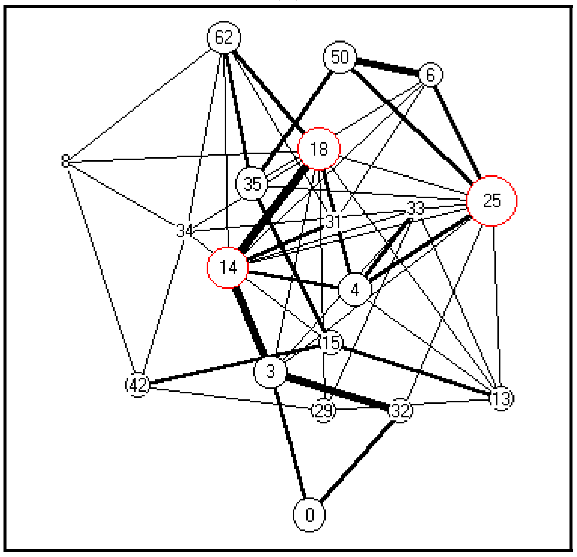
- Each visualization allowed the user to identify different properties of clustering. With the graph drawing visualization, for example, it was possible to understand the connectivity between the clusters; thick edges in this visualization also indicated pairs of clusters that could



(a)



(b)



(c)

Figure 4.3: Three visualizations of a clustering: (a) a histogram chart, (b) a scatter-plot graphic, and (c) a clustering graph drawing.

possibly be improved. The histogram provided a better understanding of how the size of the clusters varies, and offered an intuitive visualization of the satisfaction of the cluster-size constraints. The scatter-plot provided a more detailed view than the histogram, by identifying clusters for each size.

- The visualizations are intuitive media for humans to perform selection or manual changes of clusters. In the current version of the system we allowed the user to select clusters by clicking on their related circles in the graph drawing visualization. Other visualizations, however, can offer more interesting ways of interacting with a clustering solution. The system could, for instance, allow selection of a group of clusters with the same size simultaneously when a bar of the histogram chart is clicked. Another interactive feature that can be implemented is to allow the user to manually transfer a histogram bar to a different location s of X-axis of the chart; this would cause the associated clusters to be re-clustered (and possibly combined with other selected clusters) in order to create new clusters with size s.

Despite the positive aspects above, the prototype showed two major drawbacks:

- It was difficult to implement support for multiple constraints. We could not successfully develop a flexible clustering method (that allowed inclusion of new elements) that was able to solve all constraints in an effective and efficient way. In many cases, the methods struggled even with simple size constraints, for which a trivial arbitrary division of the graph in a fixed number of clusters would provide a feasible solution quickly. This problem is more critical for the Simulated Annealing, since changes in the set of constraints can demand a completely different annealing schedule. We believe that the solution for this difficulty is to use a more robust automatic method for constraint solving. A good candidate is the *Lagrangian Relaxation* [77], which can adjust the weights of constraints, similarly to what the user does via interaction with the system. Another possibility is to the implement a pool of clustering methods from the literature without any change, and then applying the methods to specific problems according to the need (for instance, bisection algorithms could be used to reduce the coupling between pairs of clusters).

- While the visualizations offered interesting clues about the quality of the clustering solutions, it was difficult to follow the improvements made by the algorithms. Even small changes of the clustering could result in big changes of the visualization. The view that was less sensitive

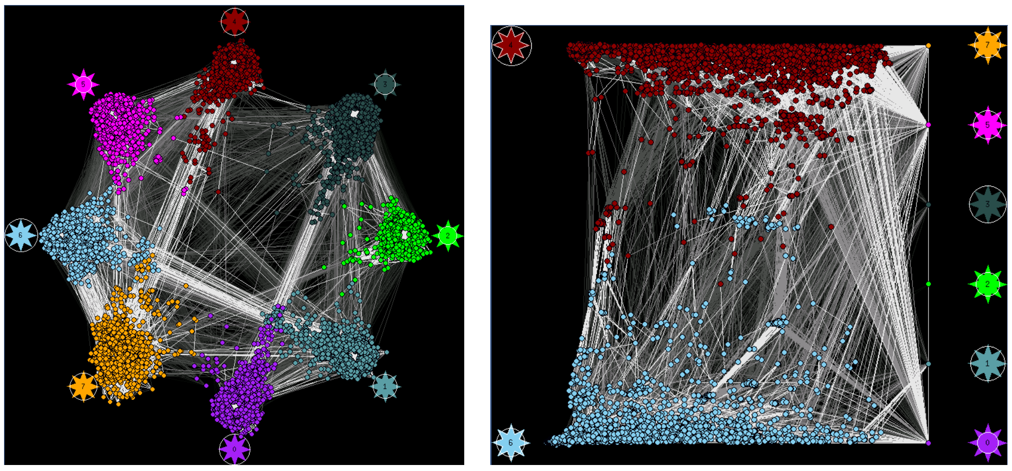

(a) (b)

Figure 4.4: A visualization of a graph partitioning solution using Springs (from [124]). Figure (b) is a focused view with only two clusters. Courtesy of Joe Marks.

to this problem was the histogram chart.

4.4.1 Related Work

An initial work on interactive graph clustering, that we encountered during the development of the User Hints prototype, was the research of Lesh, Marks and Patrignani [124] based on the HuGSS paradigm. This work was mentioned briefly in Section 2.3.3. The strongest part of the research is a novel visualization that provides the user with an impression of the cohesion between vertices of a graph and their assigned clusters. The visualization is a graph drawing produced using a variation of the spring method. The drawing is generated by modeling vertices as masses, and replacing every edge by a spring that pushes or pulls a pair of vertices according to the edge weight. Additional springs connect every vertex to a hub representing a cluster. The hubs are described graphically as stars and are distributed along a circumference. Colors are used to identify the association between vertices and clusters. Moreover, a circle around each star indicates the number of vertices assigned to that particular cluster. Figure 4.4(a) illustrates this visualization. It is possible to change the view to display only two clusters, what allows a more detailed analysis of the clustering solution; see an example in Figure 4.4(b). The user can interact with the visualization by manually reassigning a vertex to a different cluster, or running a graph partitioning method on the whole graph or on a pair of clusters.

The visualization is interesting since it provides a clue about the cohesion between vertices and their clusters. This is obtained by mapping general attributes of the clustering solution to euclidian proximities between elements of the drawing. For instance, vertices that are connected to many other vertices in different clusters may appear far away from their hubs, therefore, indicating a weak attachment to their associated clusters.

Unfortunately, human experiments done by Lesh, Marks and Patrignani with a benchmark set – the ISPD Circuit Benchmark suit [6] – did not result in significant improvements of the clustering solutions [123]. The experiments showed that users could choose valid pairs of clusters for improvement, but the overall gain was minimal when compared to the initial quality of the solutions obtained by using traditional non-interactive clustering methods. One explanation for this was that the current clustering methods available in the literature already provide very good results. A second explanation was that the benchmark data consisted of thousands of vertices, which had no meaning for the users; consequently, domain knowledge was not available to help the users find better partitions.

Since the experiments done by Lesh *et al.* indicated a limited space for improving graph clustering solutions, we decided not to do any further investigations of the User Hints framework for this problem (for example, we did not include a focus facility and we did not implement better algorithms). Rather, we moved on to two other optimization problems that are more promising for human intervention. We discuss these problems in the following chapters.

4.4.2 General Conclusions

The work presented here helped us to realize some benefits of having user hints and paved the way for the design of interactive frameworks for other problems. The main conclusions of the research are that:

- Multiple visualizations help to understand and to interact with an existing clustering solution; however, a good visualization – that not only shows the structure of the clustering, but also allows an incremental comprehension of how the solution is improved by optimization methods – is difficult to design.

- User hints are better employed in harder optimization problems, where constraints change dynamically and the user's desires are difficult to express in terms of a precise objective function, among other characteristics. For the traditional graph clustering problem, heuristics already exist that can provide good results.

The prototype that we developed serves as a toy for learning how constraints affect a graph clustering solution. More complex constraints may be included for this aim, such as forcing some vertices with a particular attribute to be placed in the same cluster. Nevertheless, finding other practical and realistic applications for the prototype may be a problem.

CHAPTER 5

User Hints for Directed Graph Drawing

In this chapter we apply the User Hints framework to the problem of drawing directed graphs. We investigate three kinds of hints: focus on a region of the drawing that needs improvement, insertion of layout constraints, and manual changes to the drawing. The framework is evaluated for the goal of helping convergence. The optimization method we use is based on the Sugiyama method.

Part of this study was presented in the Graph Drawing Conference 2001 [48] and at the Australian Symposium on Information Visualization [47].

This chapter is organized as follows. Section 5.1 introduces the Graph Drawing Problem. Section 5.2 explains the need for having user interaction in Graph Drawing applications. Section 5.3 presents our framework based on user hints, which incorporates focus and layout constraints into the Sugiyama method. An interactive graph drawing system that follows the framework is described in Section 5.4. Section 5.5 presents a pilot study of the system involving human experiments. Finally, Section 5.6 draws some remarks about the interactive graph drawing framework.

5.1 Graph Drawing

Graph Drawing is an emerging research area with strong applications in Information Visualization. The general framework used in Graph Drawing is illustrated in Figure 5.1. A graph representing relationship between a group of entities (vertices) is input to a graph drawing algorithm. The algorithm then produces a visual representation of the vertices and the edges of the graph.

The main aim of Graph Drawing is to produce "nice" visualizations, that help to understand the relationship between the vertices. For example, Figure 5.2 shows two drawings of the same graph, describing links between files accessible from the web site of the Information Visualization Group at The University of Sydney[1]. The second drawing is certainly more understandable and easier to

[1]The URL of the web site is *http://www.it.usyd.edu.au/˜visual*.

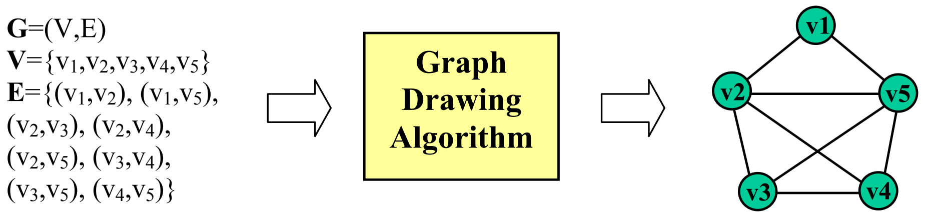

G=(V,E)
V={v1,v2,v3,v4,v5}
E={(v1,v2), (v1,v5), (v2,v3), (v2,v4), (v2,v5), (v3,v4), (v3,v5), (v4,v5)}
Graph Drawing Algorithm
v1
v2
v5
v3
v4

Figure 5.1: The general graph drawing framework.

follow than the first one.

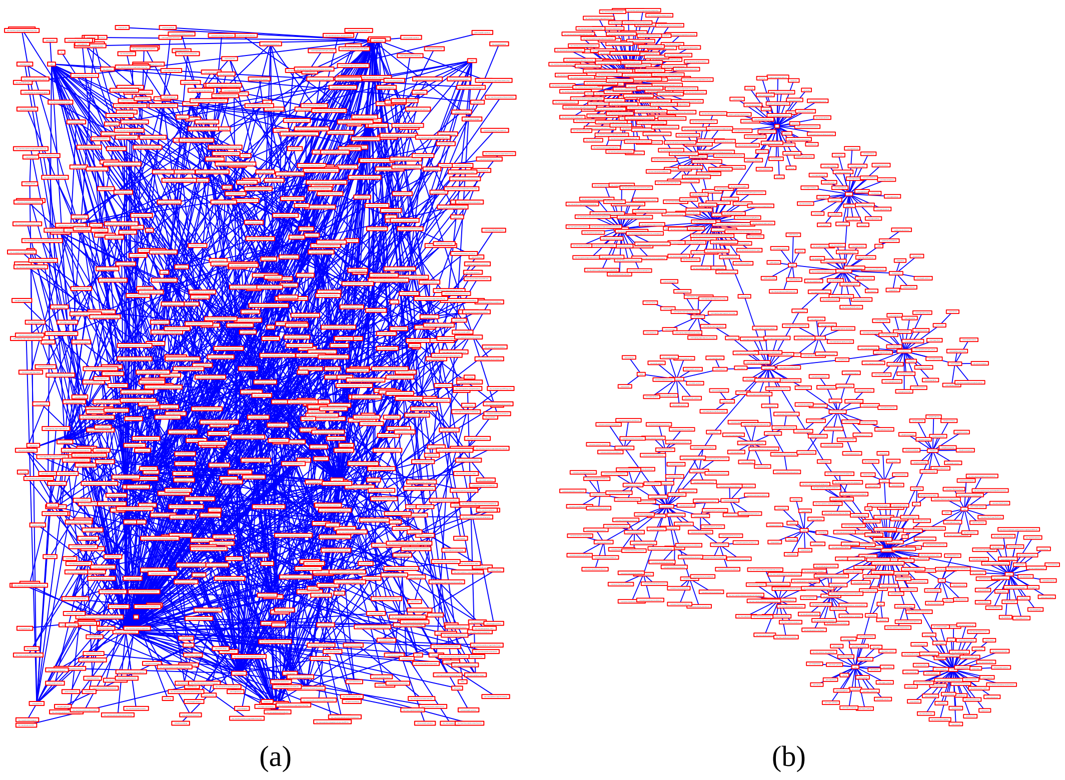


Figure 5.2: A random and a spring-based drawing of the graph structure of the Information Visualization Group web site and related pages. The graph was produced using the *LinkChecker* software (*http://linkchecker.sourceforge.net/*).

In this chapter we concentrate on directed graphs. Drawings of such graphs appear in many different domains, including science books, magazines, technical manuals, and in software applications for helping designing, managing, exploring and learning processes. When a graph contains only a few vertices and edges, it can be drawn easily by hand. However, as the number of vertices and edges increases, a manual drawing approach becomes very time consuming and difficult to manage. The solution to this problem is to make use of automatic techniques, which embed some user-desirable aesthetic criteria and apply layout algorithms to find aesthetically pleasing drawings.

A comprehensive study of aesthetic criteria and techniques for graph drawing is presented in [43]. Some aesthetic criteria commonly used for drawing directed graphs are to *show few edge crossings*, to *present few bends on the edges* (in case the edges are allowed to curve; otherwise, they are drawn as straight lines), to *display symmetry*, and to *minimize the area necessary for the drawing*. It is also desirable to *show a uniform orientation for the edges*, for example, having them pointing downward as much as possible. Even though the concept of "niceness" is inherently subjective, it has been shown that these criteria help to improve readability of the diagrams [153].

5.2 User-Interaction Suitability

Many graph drawing methods have been developed to produce drawings of graphs that satisfy aesthetic criteria, such as the ones described above. Unfortunately, the satisfaction of most aesthetic criteria usually involves NP-hard problems. The aesthetics may also conflict, that is, there may be no optimum solution for two criteria simultaneously. As a consequence, most graph drawing methods are heuristics that work reasonably fast, but may result in poor quality drawings. Even amongst papers in Graph Drawing [43, 177, 178, 182], one can find drawings that are produced in a few seconds, but present many edge crossings, edge bends or no symmetry. For instance, drawings of the Unix System Family tree that appear in many papers show at least one edge crossing [178]. 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 an automatic method for generating an initial drawing, and then improve the drawing manually. In many cases, the user can easily recognize part of the drawing that needs to be improved, and define strategies for producing a better layout. This is a common way of creating a winning drawing for the *Graph Drawing Contest* [20, 27, 28]. In general, the winners use a graph drawing method to produce a reasonably good initial layout, and manually change the automatically-generated drawing in order to satisfy additional aesthetic criteria (including aspects related to domain knowledge) that the method does not consider.

Another alternative approach is to develop better (and more complex) algorithms that consider several rules about how vertices and edges of a graph should be drawn. Meta-heuristics seem to provide a promising technique in this case, due to their flexibility in dealing with many aesthetic criteria, and ability in exploring large areas of the solution space [39, 53, 187]. The use of Simulated Annealing for drawing general graphs [38, 39] is the most remarkable example of satisfaction of several aesthetic criteria using a meta-heuristic method. However, it is not clear that better algorithms will ever eliminate the need for having users perform some post-processing improvement. There is always a graph instance for which the fast algorithms will not produce the best solution, since the space of solutions is too large to be adequately searched in a reasonable amount of time. Moreover, it is common to have many equally good layouts for the same graph, where the decision about which one to take is subjective or domain dependent. Even when some subjective aspects can be modeled as objective functions and constraints inside a flexible algorithm, it is difficult to ensure that all user preferences were considered, and that they imply no ambiguity by leading to a single "optimum" solution. Furthermore, it may be difficult to ensure that a particular algorithm is capable of finding such a solution. In the most extreme situation the user is still important for validating the result produced by an automatic method or for selecting between a number of good drawings.

The need for human intervention in graph drawing was well indicated by Donald E. Knuth in his lecture in the Graph Drawing Conference in 1996 (see Chapter 1). The report of the Graph Drawing Contest in 1999 [28] also mentions this issue and motivates the development of interactive systems. We quote here part of the conclusion of the report:

"As in past years, most of the winners combined automated and manual techniques to great effect. Given this distinct pattern in how graph-drawing software is used, it is perhaps surprising that few systems have been designed to give explicit support to this kind of human-computer cooperative design. A future graph-drawing contest may therefore include an interactive-editing category."

In the next section we propose an interactive approach for improving drawings of directed graphs by having users providing hints. User interaction is aimed at helping the method to escape from local minima and to produce good solutions directly. We consider the problem of drawing directed graphs since it appears in many real applications and involves several difficulties. The popular method of Sugiyama *et al.* [178] is used as our basic optimization method.

5.3 An Interactive Graph Drawing Framework

As described in Chapter 3 user hints should help to refine a problem and to improve convergence. In the scope of this chapter, user hints help graph drawing algorithms to search for high quality graph drawings according to a fixed set of aesthetic criteria.

Types of Hints

We consider three kinds of hints for directed graph drawing:

- **Focus.** The user can focus the drawing algorithms on some vertices of a drawing of a graph. In general, after running a graph drawing algorithm we get a reasonably good layout, but with some areas presenting poor quality according to a set of desired aesthetic criteria. The focus mechanism allows the user to reapply the graph algorithms only on those areas. The position of the vertices that are not focused is not changed.

- **Layout Constraints.** Layout constraints are useful for helping the system to improve 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.

- **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 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 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 toward a better solution.

The User Hints framework for graph drawing is shown in Figure 5.3. Arrows with a capitalized label represent hints given by the user. Note that all types of hints are direct or indirect inputs to an optimization method. The optimization method consists of improvement algorithms that work on an existing drawing of a graph.

The drawing activity is executed as follows: the optimization method automatically creates an initial working solution; then the user starts an iterative process where it is possible to perform manual changes to this drawing, specify layout constraints and re-execute the method on a focused area. When re-applied, the optimization method creates a new layout that may or may not be better than the previous one. The framework, however, keeps the best drawing computed so far by using the best solution agent as defined in Chapter 3. Every new drawing created is analyzed and compared to the best drawing. If the new drawing is better than the current best one, then the best drawing is immediately updated.

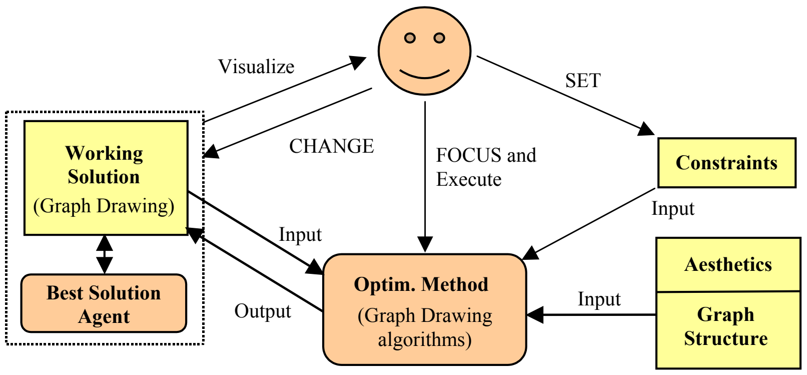


Figure 5.3: The interactive framework for Graph Drawing.

The system provides the user continuous feedback about its state by showing qualitative attributes of the current drawing.

5.3.1 The Sugiyama Method

For the purpose of drawing directed graphs, we use an optimization method based on the Sugiyama method [178]. The Sugiyama method draws a graph on a set of horizontal lines called *layers*. The method consists of four steps [43, 178]:

1. *Cycle Removal*: this is a pre-processing step that reverses some edges of the graph in order to make it acyclic. The Cycle Removal step is usually done by solving the *Feedback Arc Set problem* [43].

2. *Layer Assignment*: in this step, the vertices of the graph are assigned to layers, so that the edges show a uniform orientation (they point downwards). When an edge (u, v) spans one or more layers, it is replaced by a set of edges (u, n_1),$(n_1, n_2), \ldots (n_i, n_{i+1})$,$(n_k, v)$, where the $n_i, i = 1, 2 \ldots k$, are new vertices called *dummy vertices*. The problem of minimizing the height – and possibly the width – of the drawing is also considered here by choosing a proper layer assignment.

3. *Crossing Reduction*: In this step, the vertices in each layer are re-sorted in order to reduce the number of edge crossings.

4. *Horizontal Coordinate Assignment*: in this step, the X-coordinate of each vertex is defined so that the resulting drawing shows straight edges as much as possible, has few edge bends and is not very wide. All edges changed in the Cycle Removal step are also reversed to their original orientation in this step.

The Sugiyama Method involves several NP-hard problems [43], such as the *Feedback Arc Set problem*, in the Cycle Removal step, and the crossing minimization done in the second step. The crossing minimization is NP-hard even for only two layers [75].

5.3.2 Implementing User Hints in the Sugiyama Method

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 the vertices. Layout constraints can be defined only for real vertices, not dummy vertices. Some similar kinds of constraints for the Sugiyama method are investigated in [118] and [22].

For simplicity, in the rest of this chapter we use the term *selected vertex* to mean a vertex in the selected set A, and *fixed vertex* for a vertex in $V - A$. The drawing is constructed on an infinite grid of integer coordinates. The rows of the grid represent layers. A set of K layers is labeled $L_1, L_2, \dots L_K$, starting from bottom to top. We use the notation l_v to indicate the layer assigned to a vertex v, with $l_v \in \{L_1, L_2, \dots, L_K\}$. The way in which we implement focus and constraints is slightly different for each step of the Sugiyama method. Next, we explain this implementation in detail.

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 cycle removal). Top-Down layout constraints, however, have 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 cycle with four vertices, a, b, c and d. There are four basic ways of drawing this cycle such that the number of downward edges is maximum (optimal). These drawings are shown in the Figure 5.4(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 from a to c, then this operation reduces the number of optimal solutions to two, shown in Figures 5.4(a) and (b). If the user inserts another Top-Down constraint, from b to d, then this leaves us with a single optimal solution, Figure 5.4(a). The user can go even further by inserting another Top-Down constraint to have c drawn above b. In this case, the constraints define a precise order: a above c, c above b, and b above d, and thus the only valid solution is to reverse the edge (b, c) and (d, a), as in Figure 5.4(e).

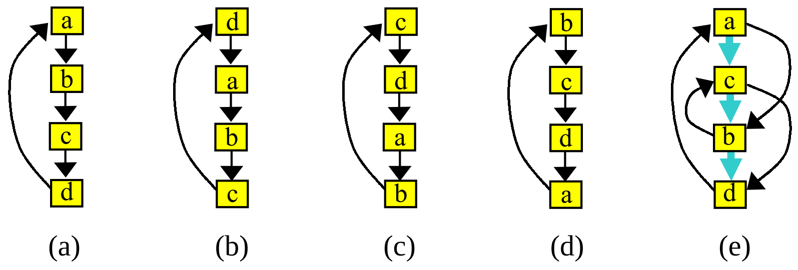


Figure 5.4: Different ways of drawing a cycle. Thick arrows represent layout constraints.

In summary, layout constraints can be used not only to reduce the number of feasible solutions, but also to force the system to consider 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) are not in $L\}$. If the resulting graph G' is acyclic then the problem is solved. Otherwise, a method for the Feedback Arc Set problem is applied to this graph, but it reverses only the original edges. For instance, merging the cycle from the previous example with the set of constraints in Figure 5.4(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 constraints. The optimal solution would be to reverse only (d, a).

We modify the *Greedy-Cycle-Removal* heuristic in [43] in order to solve the Feedback Arc Set problem with constraint edges. 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 an ordering for the vertices of G'. All edges $(u, v) \in E'$ where v appears before u in S are reversed, resulting in an acyclic graph. Our modification of the Greedy-Cycle-Removal is minor, and is highlighted in bold in Figure 5.5. The modification ensures that no constraint edge is reversed. Moreover, the algorithm can still run in linear time. Proofs for these properties of the algorithm are presented at the end of this chapter, in Section 5.6.6.

```
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 $S$.
```

Figure 5.5: The modified version of the Greedy-Cycle-Removal heuristic presented in [43].

Layer Assignment

The Layer Assignment step is executed for the graph $G' = (V, E')$ produced by the previous step. Recall that E' includes constraint edges. Focus is considered in this step, and it is implemented by modifying a well known layering algorithm so that it does not change the coordinates of fixed vertices. Note that edges with fixed vertices at both ends may not affect the layering algorithm, so they can be removed to increase efficiency. This approach is presented in Figure 5.6.

We use the Longest Path Layering heuristic [57] to construct a layering of G'. This algorithm results in drawings that are in general too wide; however, it runs in linear time and can be easily modified to handle focus.

The original version of the Longest Path Layering heuristic places all sinks in the bottom layer

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 to leave the coordinates of the fixed vertices unchanged.

Figure 5.6: The approach for Layer Assignment with support to constraints and focus.

L_1. Then each remaining vertex u is placed in layer L_{p+1}, where p is the length of the longest path from u to a sink. In our modified version we extend the procedure to include also the longest path between a vertex u and fixed (non-selected) vertices. All selected sinks are placed on Layer L_1 as previously. However, fixed vertices already have Y-coordinates assigned to them. Every non-sink selected vertex u is then placed in layer L_m, where $m = max\{p + q$: there is a sink or a fixed vertex v in layer L_q, and the longest path from u to v has length $p\}$.

Note that our layering algorithm may violate Top-Down constraints in the 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 fixed vertex or a selected sink, 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 by the algorithm. If (v_1, v_2) is a constraint edge, then this constraint is violated since v_2 is assigned to a layer above v_1. All other edges (v_{i-1}, v_i), $i = 3, \ldots, k$ point downward.

We developed a solution for the case where v_k is a selected sink. It consists of adding a post-processing step that uses the previously computed layering to shift some vertices down. Basically, for each non-source selected vertex $u \in V$ taken in the topological order, we reassign u to a new layer $l_u = min\{l_v - 1$: for all vertices v such that there is an edge $(v, u) \in E'\}$.

The revised algorithm still runs in linear time. It moves all vertices $v_2, \ldots, v_k$ downward by $k + l_{v_k} - l_{v_1} - 1$ layers. 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 places focus at a higher priority than layout constraints.

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 [178] 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 5.7.

```
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
           neighbors in $L_{i+1}$, then its original 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
           neighbors in $L_{i-1}$, then its original position is preserved.
        b. FixConstraints ($i$).
```

Figure 5.7: The barycenter algorithm for Crossing Reduction with support to constraints and focus.

The algorithm uses a heuristic called *FixConstraints(i)* that reorganizes the selected vertices in layer L_i, so that every selected vertex occupies a unique integer position in the layer and the number of unsatisfied Left-Right constraints is minimized. The heuristic is given in Figure 5.8. It has complexity $O(m(n+W))$, where m is the number of selected vertices in layer L_i, n is the total number of vertices in this layer, and W is the width of the layer in the current drawing.

```
FixConstraints(i)
Let G=(V,E) the graph to be drawn, and $Wl$ and $Wr$ the $X$-coordinates of the left-
most and the right-most vertices in layer $L_i$, respectively.
1. Compute a list $S$ of all selected vertices of $G$ in layer $L_i$.
2. Sort $S$ in non-increasing order of $X$-coordinates.
3. While S is not empty do
    a. Let $u$ be the first vertex in $S$ (that is, $u$ is the left-most vertex in $S$), and
       let $x_u$ be the current $X$-coordinate of $u$.
    b. Choose an integer number $p$ in $[Wl-1, Wr+1]$, such that no vertex in $V$-$S$
       that is in layer $L_i$ has $X$-coordinate $p$, and the number of unsatisfied
       Left-Right constraints is maximally reduced by moving $u$ to position $p$.
       If there is more than one integer $p$ that gives a minimal number of
       unsatisfied constraints, then choose the closest value to $x_u$.
    c. Set the $X$-coordinate of $u$ to position $p$ ($x_u \leftarrow p$).
    d. Remove $u$ from $S$.
    e. Adjust $Wl$ and $Wr$ to include $p$ in the interval: $Wl \leftarrow min(Wl,p)$; and
       $Wr \leftarrow max(Wr,p)$.
```

Figure 5.8: The *FixConstraints* heuristics.

Left-Right constraints may involve vertices in different layers. If this is the case, all constraint edges that have at least one vertex in layer L_i will be taken into consideration. Note that *FixConstraints* does not guarantee to solve all Left-Right constraints, since it analyzes the layer locally, and it demands the existence of empty positions for moving vertices. Some vertices may also be fixed, preventing constraint resolution. Nevertheless, the heuristic solves most constraints when applied iteratively by the barycenter algorithm.

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 result in many bends or long edges. Furthermore, the algorithm in Figure 5.7 can be re-applied by the user for improving the horizontal coordinate assignment of vertices that cause bends or long edges.

5.4 The GDHints System

We implemented the Sugiyama steps described in the previous section in an interactive system, called GDHints[2]. A snapshot of the system is shown in Figure 5.9. The system includes:

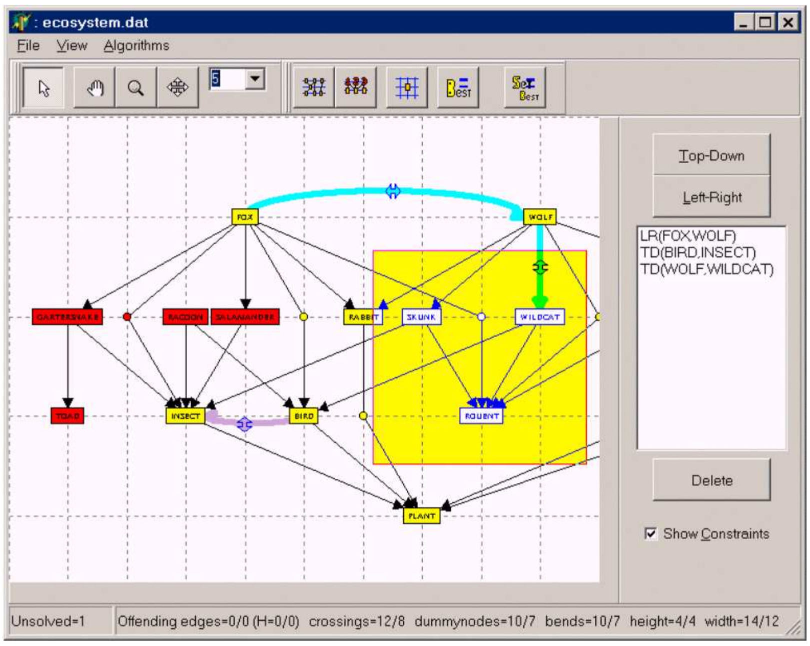
- a user interface, by which the user can select vertices for focus, add and delete constraints or perform manual changes;
- graph drawing functions for layering (cycle removal and layer assignment) and ordering (for crossing reduction); and
- displays of quality metrics of drawings.

User-System Cooperation and Quality Feedback.

The system and the user work together to improve a drawing of a graph $G = (V, E)$. The drawing is improved when its new layout is better than the previous one according to the following priority of aesthetic criteria:

- showing few *offending edges*; an offending edge is an upward or a horizontal edge;
- showing few edge crossings;
- showing few dummy vertices;

[2]The GDHints system is included in the CD-ROM attached to this thesis. See Appendix B for more details.



LR(FOX,WOLF)
TD(BIRD,INSECT)
TD(WOLF,WILDCAT)

Unsolved=1 Offending edges=0/0 (H=0/0) crossings=12/8 dummynodes=10/7 bends=10/7 height=4/4 width=14/12

Figure 5.9: GDHints – an interactive Graph Drawing system based on user hints.

- showing few edge bends (a bend occurs when an edge changes its direction on a dummy vertex); and
- minimizing the drawing area.

At the beginning of the processing, the system produces an initial drawing by assigning layers and X-coordinates to the vertices of the graph inside an area of size $|V/2| * |V/2|$. The system then selects all vertices of the graph and calls the layering and the ordering functions to create a better initial drawing.

After this first stage, the user can interact with the system. It is possible to call the drawing functions again for redrawing selected parts of the graph, specify layout constraints, and manually move vertices in the drawing.

The system evaluates the quality of every new drawing 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 to the user by showing a picture of the drawing being improved and textual information about its quality. The quality of the best solution saved by the system is also shown for comparison with the current drawing. In addition, the feedback includes colors for highlighting bad quality aspects of the drawing, and sound and animation events for calling the user's attention whenever a new and better solution is created.

5.5 Pilot Study

We did an initial study of the GDHints system with human subjects. The aim of the study was to verify whether users could improve drawings of graphs by applying the interactive tools in our framework.

5.5.1 Experiment Setup

Five subjects took part in the study: a Ph.D. student, an honors student, a research assistant, and two post-doctorate visitors, all of them from the Information Visualization Group at the University of Sydney. The subjects had a good knowledge of Graph Drawing, but were not involved in the development of the User Hints framework; therefore, a 30-minute introduction to the GDHints system was provided before starting the experiments.

The study involved three kinds of experiments:

- Experiment *E1* (constraints only): the system allowed insertion and deletion of layout constraints; manual changes and focus were deactivated. The vertices of the graph were automatically selected by the system when the user called the graph drawing functions.
- Experiment *E2* (constraints + focus): the subjects could insert and delete constraints, as well as select vertices for focusing.
- Experiment *E3* (constraints + focus + manual changes): the complete functionality of the system was available.

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 intuitive features (the subjects tend to use this option mainly); therefore, we considered them only in experiment *E3*.

Six graphs were used in the study, details of which are shown in Table 5.1. Graph *G1* is a social network. Graphs *G2*, *G3*, *G4*, *G5* and *G6* are from graph drawing papers. Graph *G2* is the proper K-layer graph from [182]. Graph *G3* is a predator-prey ecosystem, shown in [58]. Graph *G4* is based on the C-language syntax graph; it has one more vertex (labeled "v8") and an extra edge

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	0	24
G3 – Ecosystem	15	26	0	16	7	7	48
G4 - Csyntax	34	46	4	12	39	19	182
G5 – Unixsys	41	49	0	4	24	9	132
G6 - Worlddyn	43	69	6	70	144	56	360

Table 5.1: Graphs used for the experiments with the GDHints system.

than the original graph shown in [177, 178]. Graph *G5* is from paper [89] and represents the Unix System family-tree. Graph *G6* is the Forrester's World Dynamics graph, taken from [182]. Some of the graphs have vertices labelled by numbers, while others have proper names. Table 5.1 gives the numbers of vertices ($|V|$) and edges ($|E|$) of the graphs, and quality parameters for the initial drawings created by the system.

The order in which the experiments were carried out was the same for all subjects: the three types of experiments (*E1*, *E2* and then *E3*) were done for each graph at a time. We started with small graphs, leaving the largest ones to the end, so that the subjects could improve their skills smoothly. In total 90 experiments (5 subjects x 3 types of experiments x 6 graphs) were done. We allowed 20 minutes for each experiment.

In the introduction session, the subjects were informed about some possible improvements for graphs *G4*, *G5* and *G6*. For instance, they were told that graph *G5* allows a downward planar drawing. The aim was to motivate the subjects for the experiments.

The users' actions were recorded in history files. The history contains a detailed list of all operations executed by the subject, the drawings produced by each operation, and the changes on the quality of the drawings. After the experiments, the users were interviewed. We asked questions about the difficulty or facility in using the system, and their general strategy for working on the drawings. The comments collected from the interview were compared with the data in the history files in order to identify some relationship between the users actions and the changes in the quality of the drawings. This analysis also provided feedback about strong and weak points of our approach.

The first 15 experiments (related to graph *G1*) were not included when averaging the results, since we considered the subjects were still learning how to use the system during that time.

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 5.2: Quality of the best drawings produced by the users for all graphs.

5.5.2 Results

Table 5.2 shows the results of each experiment for the six graphs. It presents the minimum, the maximum and the average values of the aesthetics criteria for the best drawings produced by the five subjects.

Compared to the initial solutions described in Table 5.1, the number of offending edges was not improved much. Only some small improvements were achieved for graph *G4*. This is because the layering algorithm already produces results 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 subjects could solve crossings were the ones based only on constraints. When focus and manual changes were allowed, all five subjects produced better drawings in this aspect.

The numbers of dummy vertices and bends, and the area of the drawings 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. In order to eliminate edge crossings, the user may need to expand edges and define a more complex routing for the dummy vertices, which very often causes bends, creates new dummy vertices, and increases the area of the drawing.

An overall comparison of the results obtained in the three types of experiments with the quality of the initial drawings are shown in Table 5.3. The table contains, in percentage, the average values

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 5.3: Overall results of the experiments compared to the quality of the initial drawings.

from Table 5.2 divided by the initial values (from Table 5.1), 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 increase in the initial figures for these criteria. The values for offending edges are not included in the table because there was no change.

The results in Tables 5.2 and 5.3 demonstrate a monotonic improvement in the quality of the drawings when more interactive tools are used in the experiments. Unfortunately, the improvement was also affected by learning effects. Two subjects mentioned that the experience gained with experiments *E1* and *E2* helped to perform tasks in the following experiment, *E3*. Basically, the subjects had an idea about how much improvement could be done in the drawing and where changes were necessary.

An interesting point to note is the significant gap between the experiments *E1* and *E2*. Adding the focus facility to the system greatly improved the results. We considered this together with the comments from the interviews. The subjects confirmed that it was difficult to improve the drawings in the experiment *E1*, since adding new Left-Right constraints very often caused many new problems. A typical example was when the user inserted a Left-Right constraint between two vertices graphically close to each other, aiming to swap them. The solution found by the system satisfied the constraint, but moved the vertices to distant positions, thus creating new edge crossings. We concluded that the system was able to find solutions that satisfy Left-Right constraints in almost all cases, however, not the ones with minimal number of crossings and that preserved the mental map. Further, layout constraints could not be used to control the position of dummy vertices properly, since they were available only for real vertices of the graph.

The usage (as percentage of user interactions represented by mouse clicks) of the main tools in the system is presented in Table 5.4. The column *Select* represents the selection of elements of the drawings, necessary before moving a group of vertices simultaneously, or before running the layering and the ordering algorithms. The column *Other* includes less important operations such as

Exp.	Operations								
	Select	Move	Layering	Ordering	Return to Best Sol.	Add Constr.	Delete Constr.	Other	All
E1	0%	0%	10.7%	76.3%	2.4%	7.7%	2.5%	0.4%	100%
E2	16.2%	0%	11.3%	64.6%	1.9%	4.5%	1.1%	0.4%	100%
E3	15.5%	33.3%	6.9%	39.1%	2.4%	0.6%	0.1%	2.1%	100%
Total	10.6%	10.7%	9.7%	60.2%	2.2%	4.3%	1.3%	1%	100%

Table 5.4: Usage of the interactive tools of the GDHints System.

aligning vertices to the grid, zooming in, and zooming out. The row *Total* shows the overall results for all types of experiments.

Table 5.4 shows that constraints played a less important role in the optimization processing than focus (*Layering* and *Ordering* operations) and manual changes (*move operations*). Ordering was the most significant tool. 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, which is explained in the Remarks section of this chapter.

Figure 5.10 illustrates the improvement of edge crossings over time (in seconds). The chart was generated from the history file of a particular subject for the drawing of graph *G6*, the Forester's World Dynamics Graph, in experiment *E2*. All operations repeated consecutively in the file were compacted into a single action for simplification of the figure. The peaks in the chart represent user actions that resulted in worse drawings.

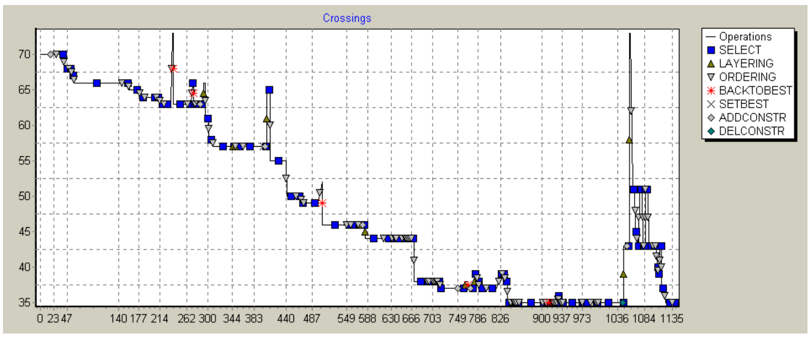


Figure 5.10: Changes in the number of edge crossings caused by user actions.

Note that the pilot study involved a limited number of subjects and, therefore, our findings can not be generalized to guess the performance of new users. We can only infer about the performance of the five subjects, since they completed a sufficiently large number (eighteen) of drawing tasks. The major result of the pilot study is that the potential of the Sugiyama Method was extended by supporting focus and layout constraints. The subjects were able to improve drawings of graphs by

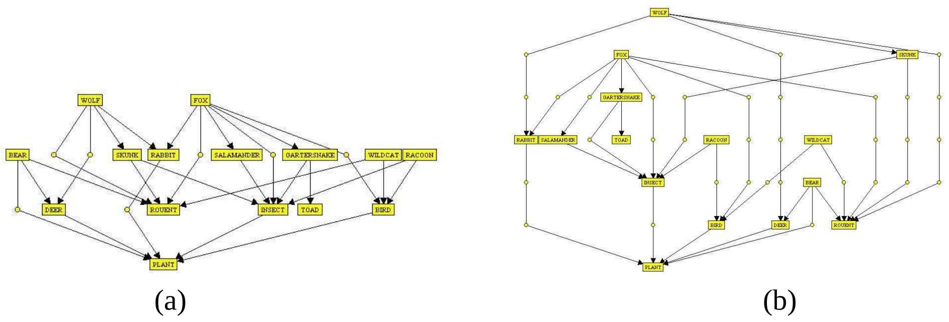


Figure 5.11: Drawings of the predator-pray ecologic system, graph *G3*. Drawing (a) has 16 crossings, 7 dummy vertices, 7 bends, width 12 and height 4. Drawing (b) has 5 crossings, 33 dummy vertices, 17 bends, width 15 and height 7.

using these interactive tools and performing manual changes. Moreover, strong and weak aspects of the framework were identified:

- Focus was a very important tool for helping the method to converge to good drawings.
- The combination of focus and manual changes offered more control of the optimization process and, therefore, increased the chance of producing better drawings.
- The attempt to use layout constraints for improving the drawing solutions was not successful.

A final observation about the experiments regards processing time. The users spent 14 minutes on average in each experiment. However, just 10% of this time was used by the system for some processing. In the other 90%, that we consider idle time, the system was waiting for the user to perform some action. During the idle 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 enhancing the cooperation between the system and the user. A new promising collaborative approach would be having the system working in the background, improving areas of the drawing selected by the user.

Examples of Drawings

Figures 5.11 to 5.14 show some drawings from experiments. The drawings labeled *(a)* are the initial solutions. These are automatically created by the system in a few seconds, and are the same for all the five users and for the three types of experiments. The drawings labeled *(b)* are examples of improved solutions created by the users.

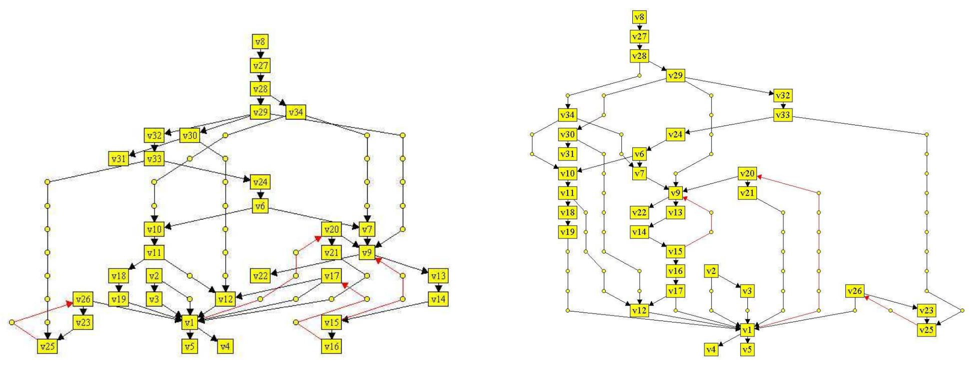


Figure 5.12: Drawings of the C language syntax graph. Drawing (a) has 4 opposite edges, 12 crossings, 39 dummy vertices, 19 bends, width 13 and height 14. Drawing (b) has 3 opposite edges, 4 crossings, 61 dummy vertices, 31 bends, width 15 and height 18.

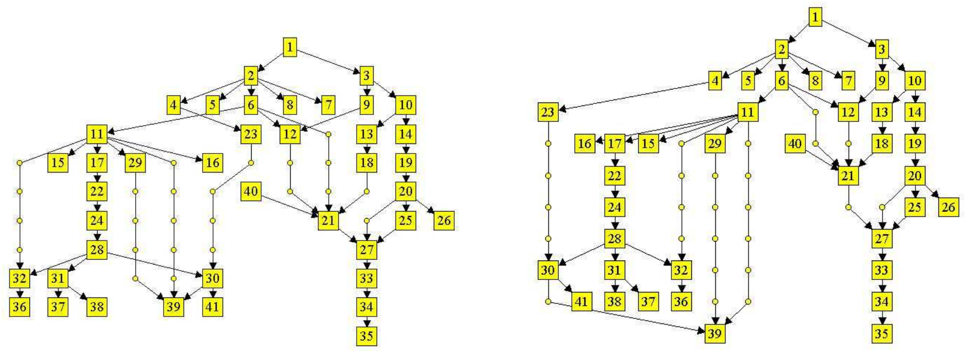


Figure 5.13: Drawings of the Unix family-tree graph. Drawing (a) has 4 crossings, 24 dummy vertices, 9 bends, width 12 and height 11. Drawing (b) has no crossing, 25 dummy vertices, 7 bends, width 13 and height 11.

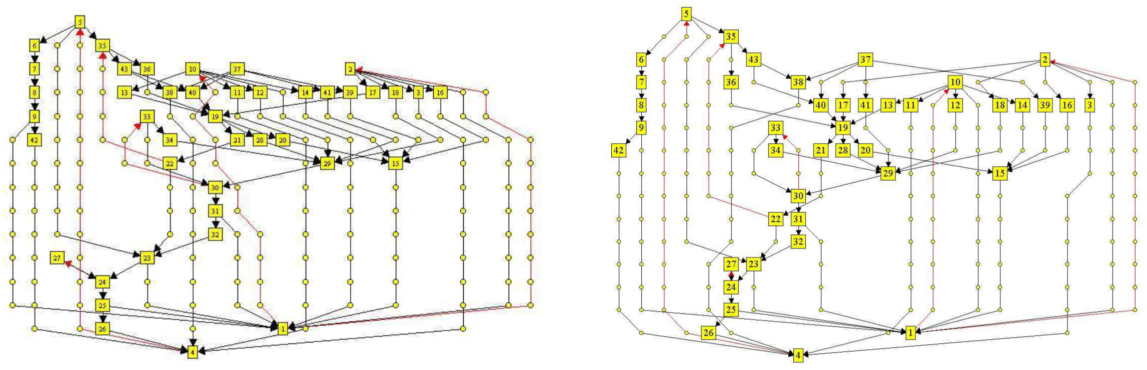


Figure 5.14: Drawings of the Forrester's World Dynamics Diagram. Drawing (a) has 6 opposite edges, 70 crossings, 144 dummy vertices, 56 bends, width 24 and height 15. Drawing (b) has 6 opposite edges, 35 crossings, 162 dummy vertices, 55 bends, width 24 and height 16.

5.6 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.

5.6.1 Constraints

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

In fact, the users found it hard to improve drawings using only constraints (in experiment *E1*). We observed that inserting a constraint and allowing the system to redraw the whole graph can result in a new drawing that is very different from what is expected. The constraint can move nearby vertices apart from each other, and can create several new edge crossings. As a consequence, the final drawing may look worse than the initial one. We believe that this problem can 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
- extending 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 them, thus destroying the mental map of the drawing. There are some mechanisms for preserving mental map explicitly. One solution is to assign degrees of movement to distinct vertices; the higher the degree, the farther the vertex can be moved. Based on this idea, vertices with many unsatisfied user-constraints would be set with the highest degree of movement. In addition, 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 [22], and it is mentioned briefly in Section 5.6.7.

5.6.2 Focus

Experiment *E2*, based on focus and constraints, demonstrated a considerable reduction in 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. Furthermore, 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.

5.6.3 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 neighborhood (that is, 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 improvement, the user can backtrack to the starting 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 iterations, 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 starting 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 the working solution as the starting point (that is, the best solution). This allowed the user to concentrate the search on the promising drawing, rather than on the previous best drawing.

5.6.4 Quality Feedback

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

- They indicated that the action made by the 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 animations and sound may help optimization tasks. There is much potential for developing better feedback strategies. One possible idea is to build intelligent agents that help to identify promising areas of the problem and/or indicate whether the user is on the right path towards a good solution.

5.6.5 Task Division

The advantage of using an interactive framework for optimization 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. This frees the user from having to analyze the quality of every new drawing, so that he or she can focus on the improvement task.

The fact that the system was idle most of the time suggests that there is space for improving the collaboration.

5.6.6 Properties of the Greedy-Cycle-Removal Heuristic

In this section we prove some basic properties of our modified Greedy-Cycle-Removal heuristic, presented in Figure 5.5. The heuristic is aimed at computing a list S of vertices of a graph $G = (V, E)$, that defines an ordering for the vertices in V. The set E consists of normal edges and Top-Down constraint edges.

Theorem: The modified Greedy-Cycle-Removal heuristic satisfies Top-Down constraint edges in E. In other words, for every constraint edge $(u, v) \in E$, v always appears after u in the resultant list S – assuming that these constraints do not conflict with each other.

The proof of this theorem is by contradiction. Firstly, however, we explain the algorithm in more details:

- Line (a) moves all sinks to the beginning of S_r.
- Line (b) moves all sources to the end of S_l.
- Line (c) moves a vertex that has no incoming constraint edge to the end of S_l; such vertices always exist when the graph is not empty, and the constraint edges do not conflict.

Suppose that the algorithm violates a Top-Down constraint (u, v). Since the relative order between vertices in S_l and S_r is never changed once they have been inserted in these lists, we can stop the execution of the algorithm immediately when a constraint is violated, and analyze the situation. Basically, there are only 6 cases that can cause the violation:

i. Vertex u is added to the beginning of S_r at line (a), when v has already been inserted into S_l. This means that v was not a source when it was added to S_l, and that it had a incoming constraint edge. However, in this case, v could not be inserted in S_l at either line (b) or (c).

ii. Vertex v is added to the beginning of S_r at line (a), when u is already in S_r. But then u was not a sink before it was inserted in S_r, and consequently it could not be added to S_r at line (a).

iii. Vertex u is added to the end of S_l at line (b), when v is already in S_l. This is similar to case (i), which shows by contradiction that v could not be in S_l.

iv. Vertex v is added to the end of S_l at line (b), and u has already been inserted into S_r. The same reasoning for case (ii) is valid here, showing that u could not be in S_r.

v. Vertex u is added to the end of S_l at line (c), and v is already in S_l. This is similar to case (iii).

vi. Vertex v is added to the end of S_l at line (c), and u has already been inserted to S_r. This is similar to case (iv).

All these cases imply contradictions regarding the way that the algorithm works; therefore, no constraint is violated.

Theorem: The time complexity of our modified algorithm is linear in the size of the graph $G = (V, E)$, $O(|V| + |E|)$, if E has no duplicated edges.

To prove this theorem we use the same data structure proposed in [43]. An array with $2|V| + 1$ buckets is created, with indices ranging from $1 - |V|$ to $|V| + 1$. Each bucket consists of a double-linked list of vertices. The array is initialized before executing the algorithm as follows: all sinks are placed in bucket $1 - |V|$, and all sources are placed in bucket $|V| - 1$; every other vertex v is inserted into bucket *outdeg*$(v)-$*indeg*(v). Each vertex v has an integer attribute $\alpha(v)$ that indicates its number of incoming constraint edges. When inserting a vertex u into a bucket b, we check whether u has an incoming constraint edge. If it does, then we put u at the beginning of b; otherwise, we insert it at the end of b. Finally, an integer variable p, used as a pointer, is set to the largest non-negative index of a bucket containing a non-source vertex with no incoming constraint edge. If such a bucket does not exist (for example, all vertices with no incoming constraint edges are in buckets with indices smaller than zero), then p is set to zero. We assume here that *outdeg*(v) and *indeg*(v) (describing the numbers of outgoing and incoming constraint edges) are variables assigned to every vertex $v \in V$, and that their were previously computed.

This initialization can be done in time $O(|V| + |E|)$.

The removal of a vertex u from the graph is executed by a routine called *REMOVEVERTEX(u)*. We consider that the following statement is true before and after the execution of the routine:

Statement *ST1*: the variable p points to the right-most bucket with index greater or equal to zero that contains a non-source vertex $v \in V$, such that $\alpha(v) = 0$. If such a bucket does not exist, p is zero. The variable p does not point to buckets with sinks or sources.

This statement is true after the initialization, and is maintained by a search operation in the routine *REMOVEVERTEX*, as follows:

1. Vertex u is deleted from its bucket. This can be done in constant time.

2. For every edge (u, v) make *indeg*$(v) \leftarrow$ *indeg*$(v) - 1$; for edge (v, u) make *outdeg*$(v) \leftarrow$ *outdeg*$(v) - 1$. This step is executed in $O($*indeg*$(u)+$*outdeg*$(u))$ time.

3. For every adjacent vertex v of u is moved from its current bucket to another bucket. If *outdeg*$(v) = 0$ or *indeg*$(v) = 0$ then we put v into bucket $1 - |V|$ or $|V| - 1$ respectively; otherwise it is put into bucket *outdeg*$(v)-$*indeg*(v). The attribute $\alpha(v)$ is also properly updated. Moreover, v is inserted at the beginning or at the end of the new bucket according to

$\alpha(v)$, as in the initialization step. This entire operation uses time $O($*indeg*$(u)+$*outdeg*$(u))$.

4. Let Q be the set of all vertices v adjacent to u, such that *indeg*$(v) \neq 0$ and *outdeg*$(v) \neq 0$ and $\alpha(v) = 0$, after the changes done in the previous step. If Q is empty then set $q \leftarrow 0$; otherwise, make $q \leftarrow$*Max*$(0,$*Max*$\{$*outdeg*$(v)-$*indeg*$(v) : v \in Q\})$. The set Q and the value of q can also be computed in time $O($*outdeg*$(u)+$*indeg*$(u))$.

5. If $q > p$, we make $p \leftarrow q$. This satisfies the statement *ST1*. If $q = p$ then the statement is already valid However, if $q < p$, then we have to update p. We know that $p \geq 0$. We perform a search for a vertex v such that $\alpha(v) = 0$, starting from the bucket of index p to the bucket of index 0. We stop the search as soon as we find a bucket with such a vertex or when we reach the bucket of index 0. The variable p is set to the index of the bucket where the search stops. Note that after this operation p satisfies the statement *ST1*. No search is necessary on buckets of index greater than p, because $q < p$ and the statement was true when *REMOVEVERTEX* was called. The search demands p steps at most. We need to perform the search because p was pointing to a bucket with a vertex w that satisfied the statement, but this vertex may have been moved by the operations executed at lines (1) and (3) of *REMOVEVERTEX*. We have four cases to analyze:

   $i.$ Vertex w is still in bucket with index p. Then the search stops immediately, without changing p, and the statement is satisfied. This takes constant time.

   $ii.$ Vertex w was moved to a bucket with index smaller than p. This happens if there is an edge (w, u). Since there are no duplicated edges, w could only be moved one bucket to the left. Then the search checks only 2 buckets (with indexes p and $p - 1$) at most, and finds w again, or another vertex with no incoming constraint edge. In total, the search is done in constant time. Note that the case where vertex w is moved to a bucket with index greater than p is not possible here, since this would imply the condition $q > p$, and it would have been treated previously.

   $iii.$ Vertex $u = w$. Then the search uses time $O(p)$=$O($*outdeg*$(u)-$*indeg*$(u)) \leq$ $O($*outdeg*$(u)+$ *indeg*$(u))$ at most. In this case, the search time is proportional to the degree of u.

   $iv.$ Vertex w was moved to the bucket of index $1-|V|$ or of index $|V|-1$ at line (3), because it is turns into a sink or a source respectively with the removal of u. Let $outdeg'(w)$ and

$indeg'(w)$ be the numbers of outgoing and incoming edges of w before the update at line (3). Note that once a vertex turns into a sink or a source, it never appears again in the "middle buckets" (with indices $2-|V|$ to $|V|-2$), and it is always deleted in this or in the following loop of the Greedy-Cycle-Removal heuristic. Consequently, we have that the time for proceeding the search is in the worst case $O(p)$=$O(\textit{outdeg}'(w)-\textit{indeg}'(w)) \leq O(\textit{outdeg}'(w)+\textit{indeg}'(w))$. But this means that it is bounded by the degree of a vertex that will certainly be deleted in the next loop, and will never appear again in a bucket analyzed by the search.

6. Delete u from the graph. This also uses $O(\textit{indeg}(u)+\textit{outdeg}(u))$ time, for removing the edges connected to u.

In the worst case, the time used by *REMOVEVERTEX*(u) is $O(\textit{outdeg}(u)+\textit{indeg}(u))+O(\textit{outdeg}(w)+\textit{indeg}(w))$, where w is a vertex that will be removed in this or in the next loop of the Greedy-Cycle-Removal algorithm.

Now, we show how we use the data structure and the routine described above in the main lines of our algorithm:

1. Line (a) calls *REMOVEVERTEX*(u) and prepends u to S_r, for every sink $u \in V$.

2. Line (b) performs a similar task, by calling *REMOVEVERTEX*(u) and appending u to S_l, for every source $u \in V$.

3. Line (c) is more complex. It must check whether $p > 0$. If the answer is 'yes', then p points to a bucket that contains a vertex u the with greatest $\textit{outdeg}(u)-\textit{indeg}(u)$ and which has no incoming constraint edge, because of the statement *ST1*. This is the vertex that we are interested in, and we call *REMOVEVERTE*(u) to remove it. However, if $p \leq 0$, then we have to perform a search for a valid vertex u for removal. We know that such a vertex exists because the graph is still not empty and that there is no conflict between constraint edges. Also, the vertex u is not in a bucket with index greater than zero, since *ST1* would imply that $p > 0$. Therefore, the vertex is in a bucket with index $i \leq 0$. Searching for this vertex demands checking $|i| + 1$ buckets. We have that $|i| + 1 = |\textit{outdeg}(u)-\textit{indeg}(u)| + 1 \leq \textit{outdeg}(u)+\textit{indeg}(u) + 1$. Thus, to find u we use time $O(\textit{outdeg}(u)+\textit{indeg}(u))$. We then call *REMOVEVERTEX*(u). The vertex u is appended to S_l once it has been removed from G.

In the worst case, the degree of every vertex u in G is counted a constant number of times in the total time involving the main lines of the algorithm: (1) during the search operation in line (c); (2) when u becomes a source or a sink by removing another vertex, and p was pointing to a bucket that contained u – this is case (iv) of the routine *REMOVEVERTEX*, which demands a search with time proportional to u; and (3) finally when u is removed.

Concatenation of the lists S_l and S_r, and reversion of the edges of the graph that are against the ordering defined by S can be done in linear time. The time complexity of the whole algorithm, therefore, is linear.

We note that the original Greedy-Cycle-Removal heuristic is proven to produce a list S of vertices that reversers at most $|E|/2 - |V|/6$ edges, if no double edges ((u, v) and (v, u)) exist. However, we cannot guarantee this performance, with our modification.

5.6.7 Related Work

Paris [146] presents an interactive single-line diagram editor that serves as a user interface to a real-time power system simulator. The editor exploits a collaboration between user actions and automatic tools based on an adaptation of the Sugiyama method in a very similar way to our User Hints framework. The aim is also to improve drawings or to adjust them according to domain knowledge. The user can perform manual changes and run the automatic tools on a selected part of a drawing such as in the GDHints system. The editor, however, is based on a different graphical representation for the drawings, and details of how some interactive features are implemented are not given.

Böhringer and Paulisch [22] introduce an interactive constraint-based system for directed graph drawing. The user can specify layout constraints that assign vertices to particular positions or ranges in the diagram, define a relative position between vertices, and group vertices together in clusters. A drawing is created by a constraint satisfaction system integrated with the Sugiyama Method. Basically, each step of the Sugiyama method implies some automatic constraints on the X and Y positions of the vertices. These constraints are combined with the user-defined constraints, and are solved by the constraint system in order to obtain a drawing solution. If an initial drawing of the graph already exists, then special constraints with low priority are also included to ensure that the new drawing does not differ drastically from the previous one. The special constraints, for example, may define a horizontal and vertical ordering for all vertices according to their positions in the initial drawing.

5.6.8 A Better Optimization Method in the GDHints System

The optimization method in GDHints can be improved by improving the algorithms in the steps of the Sugiyama method. The heuristics that we implemented are only one possible choice. We could, instead, use a constraint satisfaction system, such as the one described in the previous section. There are also many simple heuristics that may offer promising results. Examples are swapping the position of the vertices of a layer, or using the median heuristic instead of the barycenter for crossing reduction [43].

Better optimization methods can produce drawings with higher quality. The question of interest is how to support the same type of interactive facilities discussed in this chapter in another algorithm. Moreover, it would be interesting to see whether human interaction is still necessary when using a more effective method. In the next chapter we approach these issues.

CHAPTER 6

Extending the Graph Drawing Case Study: A Focus and Constraint-Based Genetic Algorithm

In the previous section we saw that user interaction combined with heuristics based on the Sugiyama method could provide better drawings of graphs than the heuristics alone. In the present chapter we investigate this issue further for a more effective graph drawing algorithm. We describe a focus and constraint-based genetic algorithm for directed graph drawing, and show that it can still benefit from human interaction.

The design of the genetic algorithm was published in the proceedings of the Second International Conference on Hybrid Intelligent Systems in 2002 [49], and as a technical report [50].

This Chapter is organized as follows: Section 6.1 discusses the application of Genetic Algorithms to Graph Drawing. Section 6.2 provides some definitions for drawings of directed graphs. Section 6.3 presents our genetic algorithm and describes how focus and layout constraints is supported. Section 6.4 explains the integration of the genetic algorithm into the GDHints system. Section 6.5 presents a human evaluation of graph drawing tasks using the genetic algorithm. Finally, Section 6.6 discusses other issues regarding the use of the genetic algorithm and its internal structure.

6.1 Genetic Algorithms for Graph Drawing

The success in applying genetic algorithm to different optimization problems has lead to the development of genetic algorithms systems for Graph Drawing. The expectation has been to obtain better approximate results or higher flexibility in supporting complex objective functions than traditional heuristic methods.

A number of graph drawing genetic algorithms have been proposed [29, 62, 88, 116, 137, 163, 187]. Most of them are for general graphs, or for drawing directed graphs with straight-line edges. When dummy vertices are considered and the edges are allowed to bend, the problem is more difficult; Branke *et al.* [187] treats this problem.

Despite of the promised benefits in using genetic algorithms, the applications of this method for Graph Drawing have shown serious disadvantages:

- Excessive computation time is necessary even for drawing simple graphs.
- The effectiveness of the method decreases drastically as the size of the graph increases.

Thus, many of the papers cited above contain bad drawings that could easily be improved by using some standard graph drawing heuristics.

In fact, we consider that the best genetic algorithms for graph drawing are the hybrid ones[1], that combine traditional genetic operators with domain-specific heuristics. This is the case of the systems presented by Branke *et al.* [29, 187], which implement a spring algorithm and the barycenter heuristic as mutation operators. Our own experience in this direction, using multi-agent systems for drawing general and directed graphs [46, 52, 53, 175], shows that better results are yielded by a combination of different strategies. Moreover, Branke *et al.* [187] and Rosete-Suárez *et al.* [163] also investigate the use of different solution representations, where a drawing is described not by the X, Y coordinates of its vertices, but by the relative position between the vertices. The aim is to allow simple mutation operators to perform large and effective changes of the drawing. Another aim is to abstract the major characteristics of the solutions, so that drawings with no significant differences (for example, a drawing that is a shifted or a mirrored version of another layout) are considered the same. This avoids having the population of solutions overloaded with similar drawings and, consequently, losing diversity.

Although, these extensions are beneficial, the application of genetic algorithms for graph drawing is still far away from the effectiveness and efficiency desired for this problem. We therefore propose using human interaction based on hints, so that tasks that are not properly performed by a genetic algorithm can be complemented by a user, and vice-versa.

We present a genetic algorithm for this aim, which can be applied as the optimization method

[1]The name to be given to a modified genetic algorithm can arouse political feelings. Some researchers prefer always to employ the term “hybrid” to distinguish an extended genetic algorithm from traditional genetic algorithms based on binary operators. Other researchers are more radical, and emphasize this difference by using the general expression “evolutionary approaches” when referring to hybrid genetic algorithms. In this chapter, we do make this difference in the nomenclature. We describe a hybrid genetic algorithm, but we call it a “genetic algorithm” for simplicity.

in the interactive framework described in the previous chapter. The genetic algorithm necessarily supports focus and layout constraints, and can work on an existing drawing.

Another genetic algorithm that supports similar facilities is presented by Masui [132]. This genetic algorithm solves user-defined constraints that can impose an ordering for the vertices (such as the Top-Down and Left-Right constraints that we investigate), and assign vertices to particular coordinates (this would be comparable to define fixed vertices in our approach). Our work, however, is different from the approach of Masui, because we consider that many vertices can be set as fixed vertices. Thus, our algorithm tries to ignore the fixed vertices as much as possible, and concentrates action on the selected vertices. We also use a different structure to implement our genetic algorithm, and we present more advanced operators that are based on the work of Branke *et al.* [187] and on own our experience [46, 52, 53, 175].

6.2 Graph Drawing Definitions

We use the same graph drawing standards from the previous chapter, where a directed graph is drawn on an infinite rectangular grid of X, Y coordinates. The Y-coordinates represent layers, while the X-coordinates are called columns. Layers and columns are labeled with integers from bottom to top, and from left to right. Figure 6.1 shows an example of a drawing of a graph on a grid. We, however, define here a drawing in a more formal way:

A drawing $D = (V, E, M, d)$ consists of a directed graph $G = (V, E)$, a set M of dummy vertices, and a function d that assigns X, Y coordinates of the grid to every vertex v in $V \cup M$. Every dummy vertex is related to a particular edge e in E, and uniquely identifies the intersection of e with a particular layer. We use the notations $v.l$ and $v.x$ to refer to the Y-coordinate and the X-coordinate, respectively, of a vertex v in $V \cup M$. If, for an edge $e = (u, v)$ in E, $|v.l - u.l| > 1$, then e is long and has $|v.l - u.l| - 1$ dummy vertices.

The graph-drawing problem is to produce drawings that satisfy the same aesthetic criteria defined in the previous chapter. The main difference from that chapter is that we include here a new aesthetic criterion, minimizing the total edge length, which is set with a low priority. We also include the satisfaction of layout constraints directly as part of the objective function; these constraints receive the highest priority.

The interaction with a graph drawing process consists of supplying user hints. Hints are manual adjustments of the coordinates of the vertices, Top-Down and Left-Right layout constraints, and

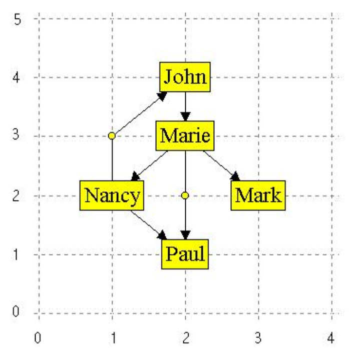


Figure 6.1: Drawing of a directed graph on a grid.

focus on a region of the drawing for improvement. Focus is performed by selecting a group of vertices. Vertices that are not selected are called *fixed*.

6.3 The Genetic Algorithm

In this section we describe a genetic algorithm that supports focus and layout constraints. The design of the genetic algorithm is based on the criterion that it should be fast and economical with memory usage.

As in the previous chapter, the selected vertices of a graph drawing are the only elements that can be repositioned. The layout of the remaining part of the drawing has to be preserved. This condition implies that having a population of individuals where each one contains a complete drawing of the graph is not an efficient approach for minimizing memory usage. Much memory is wasted, particularly when a small percentage of the drawing is selected.

At first glance, a good approach seems to be constructing an induced sub-graph with the selected vertices. A drawing could then be produced for the sub-graph independently, and combined with the coordinates of the fixed vertices. However, this is not effective because there is a strong dependency between selected and fixed vertices, mainly due to the edges connecting them. Such dependency affects – and in most of the cases restricts – the repositioning of the selected vertices. Figure 6.2 illustrates this situation. Figure 6.2(a) is an initial drawing with selected vertices d, e and f, and Figure 6.2(b) is an improved version of the drawing produced by our genetic algorithm. In order to increase the proximity between the neighbor vertices a and d, vertex d had to "jump over" the fixed vertex c. Vertices e and f also had to be moved in order to improve the layout of the edges (e, a), (f, g) and (d, f), regarding several aesthetic criteria such as minimizing the number of edge crossings and presenting uniform edge orientation. Furthermore, the selected vertices could not be placed anywhere; vertex overlap is not allowed and some positions of the drawing were already taken by fixed vertices.

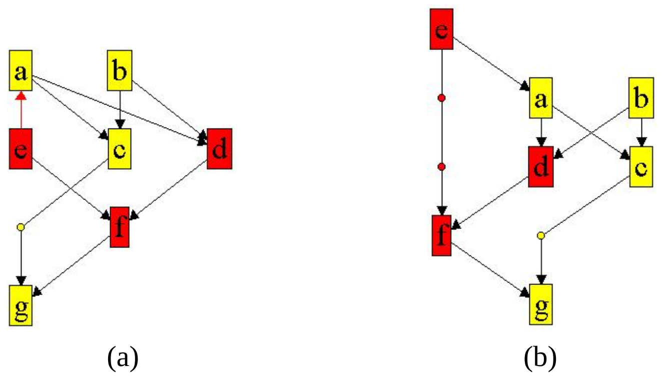


Figure 6.2: Dependence between selected and fixed elements of a graph when redrawing selected vertices.

The dependency between vertices implies that we cannot separate selected and fixed elements of the graph completely. Therefore, another way of managing selection effectively and efficiently is necessary. Our solution to this problem consists of a more complex data structure for the individuals in the genetic algorithm, and special operations that allow integration of the individuals with a complete drawing of the graph. The following section gives details of our approach.

6.3.1 Individuals

Let D be a drawing of a graph G. Let v.*selected* be a Boolean attribute that indicates, for any real or dummy vertex v in D, whether v is selected or not. We extend here the concept of selection in order to include edges: an edge e in D is selected if at least one of its endpoints or dummy vertices (if they exist) is selected. This condition is indicated by e.*selected*.

The genetic algorithm manipulates a drawing of the whole graph, a list of layout constraints and individuals. An individual contains information about only the selected vertices and the selected edges, and it can be used to reconstruct the selected area of the drawing. The genetic algorithm can have several individuals each possibly describing a different layout for the selected area. A group of individuals generated in an iteration of the genetic algorithm is called the population.

Figure 6.3 shows the structure of an individual. It consists of three main parts:

1. *REALV*: a fixed-length vector with the X, Y coordinates of all selected real vertices;

2. *EDGES*: a vector with one position for each selected edge of the drawing, holding a list of all dummy vertices associated with this edge. The list is sorted such that the dummy vertices are

in the same order as they appear when following the orientation of the edge. Every dummy vertex in *EDGES* contains its X, Y coordinate position, whether this vertex is selected or not, and a reference to the edge to which it is related.

3. *DUMMYV*: a variable-length vector with references to the selected dummy vertices in *EDGES*. The vector *DUMMYV* is used for quickly accessing the selected vertices only.

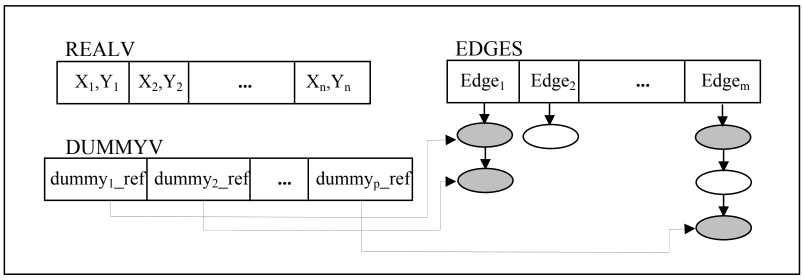


Figure 6.3: Representation of an individual. Selected dummy vertices are shown in gray color.

Note that an individual holds information about fixed dummy vertices as well as selected dummy vertices of a selected edge. In Figure 6.3, fixed dummy vertices are represented by white ellipses, while the selected ones are highlighted in gray. Although fixed dummy vertices should not be redrawn, they have to be included in the individual since they can be deleted if the length of their edges is shortened. Consider that the user has selected the endpoints of a long edge e, but not its dummy vertices. If the Y-coordinate (saved in *REALV*) of any of the endpoints of e is changed, then the length of the edge can be reduced and some of its dummy vertices need to be removed. On the other hand, if the edge length is increased, then new dummy vertices have to be created. All new dummy vertices created with this process are marked as selected.

For simplicity, we refer to a real and a dummy vertex in an individual using the same notation for vertices in a drawing, introduced in Section 6.2; that is, we use $v.x$ and $v.l$ to represent the X and the Y coordinates of a vertex v, respectively, in an individual I. If v is a real vertex, then we are referring to coordinates in the vector *REALV* of I. If v is a dummy vertex, then we refer to coordinates saved in the vector *EDGES*.

An individual can be produced directly from a drawing D of the whole graph containing a selected set of vertices. We call this operation *Extraction* and we use the notation *Extract (D)* to indicate the new individual extracted from D. As an example of *Extraction*, the drawing in Figure 6.2(a) would result in an individual with *REALV* containing coordinates for vertices d, e and f. The vector *EDGES* would have six positions – representing the selected edges (a, d), (e, a), (b, d),

(d, f), (e, f) and (f, g), each with empty lists, since there are no dummy vertices in these edges. The vector *DUMMYV* would have zero length.

It is possible to combine a drawing of the whole graph with an individual; we name this operation Merge. The notation *Merge (D,I)* represents the drawing D merged with the individual I. The merge operation updates the sequence of dummy vertices and the coordinates of all selected vertices of D according to the information in I. Note that D may already have a layout for the selected elements, which is overwritten by the information in the individual.

Extraction and Merge are used by the genetic algorithm in several stages of the graph drawing improvement. For instance, the first individual produced is extracted from an initial drawing input to the genetic algorithm. Later, new individuals are merged with this drawing in order to create a complete layout for evaluation. More details about the application of Extraction and Merge are given in the next sections.

6.3.2 Quality Evaluation

We define an evaluation function $Q(D, R) = (q_1(D, R), q_2(D), q_3(D), \dots, q_7(D))$ that measures aesthetic aspects of a drawing D, and the degree to which D satisfies a list R of constraints. This function results in a cost vector, whose parameters are as follows:

1. q_1: number of violated Top-Down and Left-Right constraints;
2. q_2: number of horizontal and upward edges;
3. q_3: number of edge crossings;
4. q_4: number of dummy vertices;
5. q_5: number of edge bends (the angle of the edges incident to a dummy vertex is not 180^o);
6. q_6: the area of the drawing (the number of layers multiplied by the number of columns used by the drawing); and
7. q_7: the sum of all edge lengths.

A priority order is defined such that q_i is more important than q_{i+1} for all $i = 1, 2, \dots, 6$.

Minimizing the number of edge crossings, bends, the area of the drawing and other aspects is a common approach for improving drawings of directed graphs. In general, low cost vectors indicate aesthetically pleasing drawings and, therefore, solutions of high quality.

Every individual produced by the system is assigned a cost vector. The cost vector measures only the selected elements of the drawing and their relation with the fixed elements. This is obtained by firstly merging the individual with the existing drawing of the whole graph, and then computing q_1, q_2, …, q_7 for the selected vertices and edges. For instance, when computing q_3 we need to count the number of edge crossings between two selected edges and between a selected and a non-selected edge. The crossings between two non-selected edges do not have to be considered for the cost vector, since they are invariant.

Given two individuals I_1 and I_2 with assigned cost vectors $Q_1 = (q_1^1, q_2^1, \ldots, q_7^1)$ and $Q_2 = (q_1^2, q_2^2, \ldots, q_7^2)$ respectively, we say that I_1 is *better* than I_2, denoted by $I_1 < I_2$, if Q_1 is smaller than Q_2 in the priority order; formally: $I_1 < I_2$ if $q_1^1 < q_1^2$, or if there is a k, $2 \le k \le 7$, such that $q_k^1 < q_k^2$ and $q_i^1 = q_i^2$ for all $i = 1, 2, \ldots, k-1$. Using this ordering, we aim to produce drawings which minimize the cost vector.

6.3.3 Evolutionary Cycle

The genetic algorithm takes as input an initial drawing D of a graph with some selected vertices, and a list R of Top-Down and Left-Right constraints. The algorithm produces a population of individuals representing different layouts for the selected elements of the drawing.

The genetic algorithm executes the following steps:

1. A copy D' of the drawing D is made. The copy, which is also a layout of the whole graph, is used in several activities including in merging.

2. An initial population P_0 is created by extracting an individual I from D' and applying the mutation operators to I several times in order to produce new individuals. The population P_0 contains the first individual I and 20 mutated clones.

3. $iter \leftarrow 0$.

4. The population P_{iter} is evaluated by computing a cost vector for every individual. Basically, for every I in P_{iter}, the genetic algorithm executes *Merge (D',I)* and computes $Q(D', R)$.

5. A sub-population S is chosen from P_{iter} by a kind of tournament selection. The best individual in P_{iter} is directly inserted into S (if there is more than one individual with lowest cost vector, then one is chosen arbitrarily). The remaining individuals in P_{iter} take part in pairwise competitions: we choose randomly two individuals I_1 and I_2 from P_{iter} that have

not yet participated in a competition; if $I_1 < I_2$, then I_1 is added to S; otherwise, I_2 is added to S. In the first generation of individuals, the competition is always between individuals created by mutation. In next generations, I_1 is chosen among the individuals created via mutation, and I_2 from the individuals created via crossover. Note that this step produces a set S with 11 individuals (the best individual plus the 10 tournament winners).

6. $iter \leftarrow iter + 1$.

7. A new population P_{iter} is created: the best individual in S (decided in step 5) is directly transferred to P_{iter}; then two individuals I_1 and I_2 are randomly chosen and removed from S. These individuals are combined by a crossover operator and are added to P_{iter}. The process repeats until S is empty. In order to complete the new population, 10 other individuals are produced by mutation of the best individual in P_{iter-1}.

8. If a stop criterion is reached, then the evolutionary cycle ends, and the genetic algorithm outputs P_{iter}; otherwise, it goes back to step 4.

In the genetic algorithm, new individuals are created through mutation or crossover of existing ones. Good individuals, with low cost vectors, are expected to succeed in the selective tournament and propagate their characteristics to the next generations. We stop the algorithm in our system after a predefined number of iterations. However, other stop criteria can be used, such as detecting when no significant improvement is made in the best individual after many consecutive iterations.

Figure 6.4 illustrates the internal functioning of the genetic algorithm.

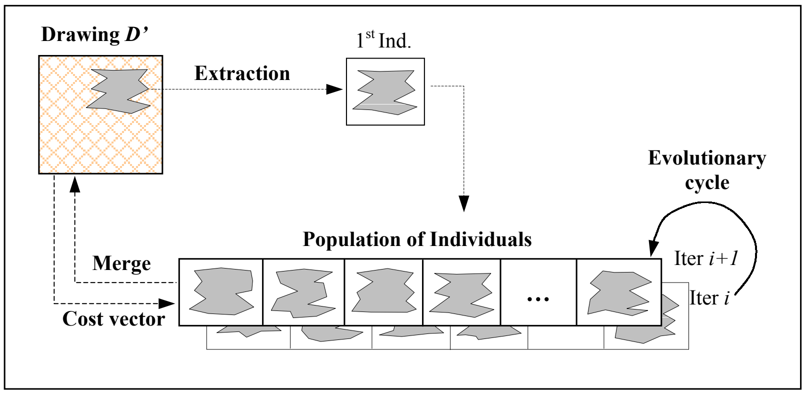


Figure 6.4: Internal functioning of the genetic algorithm.

The drawing D' is necessary since it contains the dependency between the fixed and the selected vertices and edges. An individual can be thought as a partial solution – a layout of only the selected elements. By merging this individual with D' we obtain a complete solution. The evaluation of individuals only makes sense in the context of the whole drawing.

Moreover, the drawing D' provides the position information for all fixed vertices. Such information is useful during mutation and crossover in order to avoid vertex overlaps.

6.3.4 Operators

This section describes the mutation and crossover operators implemented in our genetic algorithm. First, some basic routines are introduced.

Basic Routines

We developed five basic routines for helping to change the X, Y coordinates of selected vertices. The changes are applied to vectors *REALV*, *EDGES* and/or *DUMMYV* of an individual passed by the parameters. The routines are:

- *MOVEX (Individual I; Vertex v; Integer x).* This routine changes the horizontal coordinate of a selected (real or dummy) vertex v in an individual I. *MOVEX* tries firstly to release the position x on layer $v.l$, so that v can be moved to x ($v.x \leftarrow x$) without overlapping any vertex. For this aim, it may shift some vertices or find an alternative horizontal position for the move. If $v.x$ is already equal to x, the routine verifies whether this position is occupied by another vertex, and solves overlaps. In the next lines, we refer to vertices in layer $v.l$ only. The routine works as explained below:

    1. If there is no vertex u in position x, with $u \neq v$, then $v.x$ is set to x and the routine ends.
    2. If x is occupied by a fixed vertex, we search for the closest position x' that is not occupied by a fixed vertex (even though it may contain a selected vertex); otherwise x' is set to x.
    3. If the position x' is occupied by a selected vertex u, with $u \neq v$, then we shift the X-coordinate of u and of as many selected vertices as necessary, not including v, in order to release this position.
    4. Finally, $v.x$ is set to x' and the routine ends.

The directions for the searching and shifting actions are determined by the current horizontal coordinate $v.x$ of v, the new position x, and by the two global Boolean variables that we call *samedirsearch* and *samedirshift*. When $x < v.x$, the vertex v has to be moved to the left of its current position. However, if x is already occupied by a fixed vertex then this move is not possible and a search for a better position has to be carried out. If *samedirsearch=True*, then the routine searches on the left-hand side of the drawing starting at x; in other words: it searches for the greatest position x' smaller than x that is not occupied by any fixed vertex. If *samedirsearch=False*, then the algorithm searches in the opposite direction (looking for the smallest x' greater than x). A symmetric reasoning is adopted when $x > v.x$. When $x = v.x$, the search direction is chosen randomly. Step 3 follows a similar idea for shifting vertices, where the direction of shifting is determined by $v.x$, x, and the global Boolean variable *samedirshift*.

MOVEX can be implemented in linear time in the total number of fixed and selected vertices on layer $v.l$, if a sorted list of all vertices in this layer is provided.

- *MOVEY (Individual I; Vertex v; Integer y). MOVEY* changes the vertical coordinate of a selected vertex v in an individual I. It executes the following steps:

```
1. $v.l \leftarrow y$;
2. MOVEX ($I$,$v$,$v.x$).
```

- *UPCLOSURE (Individual I; Vertex v).* This routine returns a list of all selected vertices in layer $v.l$ or above, that can reach v, including v itself, following only edges with both endpoints selected.

- *DOWNCLOSURE (Individual I; Vertex v).* This routine returns a list of all selected vertices in layer $v.l$ or below, that can be reached from v, including v itself, following only edges with both endpoints selected.

- *MOVEYCLOSURE (Individual I; Vertex v; Integer y; Boolean upclosureflag).* This routine changes the vertical coordinate of a selected vertex v and of all vertices directly or indirectly connected to it in a particular direction. The parameter y is the new Y-coordinate to be assigned to v. The pseudo-code of *MOVEYCLOSURE* is presented below:

```
1. $d \leftarrow y - v.l$;
```

2. If *upclosureflag* then $C \leftarrow UPCLOSURE(I, v)$

   else $C \leftarrow DOWNCLOSURE(I, v)$;

3. For all u in C do

   a) $u.l \leftarrow u.l + d$;

   b) *MOVEX* (I,u,$u.x$).

Note that *MOVEX* not only moves a vertex to a new position in a layer, but also guarantees no overlap with any other vertex. *MOVEY* moves a vertex to a new layer and calls *MOVEX* to solve overlaps. *MOVEYCLOSURE* moves a vertex to a new layer as well as the vertices than can reach it or be reached from it. After executing *MOVEY* or *MOVEYCLOSURE*, all edges connected to the moved vertices are checked for invalid dummy vertices. If necessary, new dummy vertices are created and existing ones are deleted.

Although we have not explicitly mentioned here, these basic routines use the drawing D', described in Section 6.3.4, in order to query the graph structure and the fixed vertices.

Mutations

We implement three mutation operators. One is a general random mutation, one follows the well known ”barycenter” heuristic, and one solves constraints. The choice for the design of these mutations was based on the work of Branke *et al.* [187] and on our experience with the application of multi-agent systems to Graph Drawing [46, 52, 53, 175].

- *RANDOMCHANGE (Individual I)*. This operator changes the coordinates of a selected vertex in an individual I. It randomly chooses a selected vertex v, a direction to move (horizontal or vertical), and an offset $k \in Z^*$. If the direction is horizontal or if v is a dummy vertex, then the operator calls *MOVEX(I,v,$v.x + k$)*; otherwise, it calls *MOVEY(I,v,$v.l + k$)*, *MOVEYCLOSURE(I,v,$v.l + k$,False)* or *MOVEYCLOSURE(I,v,$v.l + k$,True)*. *RANDOMCHANGE* is similar to the mutation *Shake* described in [187]. The main difference is that we set k according to a linear distribution of probabilities where small absolute values of the offset have a higher chance to be chosen. Moreover, a maximum offset is defined so that the width or the height of the whole drawing is not greatly increased.

- *WALK_BC (Individual I)*. This operator follows the same idea of the mutation *Walk_BC* introduced by [187]; it implements a heuristic based on the famous barycenter method of

Sugiyama *et al.* [178]. Our version of *Walk_BC* moves only selected vertices and considers both real and dummy vertices. It randomly chooses a selected vertex v and a vertical direction $d = 1$ (for Up) or $d = -1$ (for Down). The X-coordinate of v in I is set to the arithmetic mean of all adjacent vertices of v on layer $v.l - d$. Then a vertex u adjacent to v in the direction d is chosen at random, and the barycenter process is repeated for u. The mutation continues walking until there is no more adjacent vertex to choose in the direction d. *WALK_BC* calls *MOVEX* in order to move selected vertices.

- *SOLVER (Individual I).* This operator solves an unsatisfied constraint or changes the orientation of an edge that does not point downward. *SOLVER* randomly chooses a selected vertex u that is part of an unsatisfied constraint $c = (u, v)$ or $c = (v, u)$. If c is a Top-Down constraint, then the operator calls *MOVEYCLOSURE(I,u, $v.l + 1$,False)* for $c = (u, v)$, and *MOVEYCLOSURE(I,u,$v.l - 1$,True)* for $c = (v, u)$; otherwise, if c is a Left-Right constraint, it calls *MOVEX(I,u,$v.x - 1$)* for $c = (u, v)$, and *MOVEX(I,u,$v.x + 1$)* for $c = (v, u)$. If there is no unsatisfied constraint, but the individual has an edge $e = (u, v)$ that is horizontal or upward, then the operator changes the Y-coordinate of u or v as if e were a Top-Down constraint. In the case that I does not violate any constraint and has no horizontal or upward edges, another mutation operator is called for changing I.

The mutation to be applied in order to change an individual is chosen at random.

Crossovers

The genetic algorithm has one crossover operator:

- *COMBINE (Individual I_1,I_2).* This randomly chooses a selected real vertex v that has different coordinates in I_1 and I_2, and swaps its positions in the individuals. *MOVEX* and *MOVEYCLOSURE* are used for swapping the coordinates of v.

6.4 Integration into the GDHints System

We have integrated the genetic algorithm into the GDHints system. The system provides an initial drawing of a graph using the Sugiyama method. The user then performs manual changes of the drawing, specifies layout-constraints, and/or selects some vertices for redrawing. The genetic algorithm was implemented as an independent processing thread that can be activated at any time.

When activated, it creates a copy of the current drawing being visualized by the user and starts the evolutionary cycle. Every three seconds, the application locks the population of the genetic algorithm, recovers the best individual and merges it with the current (external) drawing. The user sees a progressive sequence of changes of the selected elements of the drawing whenever the best individual is continuously improved. If the merged drawing is better than the best drawing previously saved by the system, then the new drawing is set as the best one.

Figure 6.5 shows drawings produced by the GDHints system. The drawing on the left-hand side is the an initial layout generated using the Sugiyama method. Selected vertices are highlighted in a darker color. The drawing on the right hand-side is an improved solution, produced by the genetic algorithm in 12 seconds (on a Pentium III 760Mhz with 284Mb of RAM running Windows Me).

The genetic algorithm works on any group of selected vertices. The user can select just a single real vertex, a dummy vertex of an edge, or all vertices of the graph. The selected vertices do not have to be adjacent in the graph or even in the drawing. Figure 6.6 shows improvements of two disjoint regions of a drawing.

A point to be noted is that the user cannot make manual changes while the genetic algorithm is running. This is for efficiency. If such changes were allowed, then the cost vector of all individuals in the current population would need to be recomputed, and the list of dummy vertices may have to be reconstructed in order to keep them consistent. This processing is very time consuming; therefore, we decided simply to have the users stop the genetic algorithm before performing manual changes.

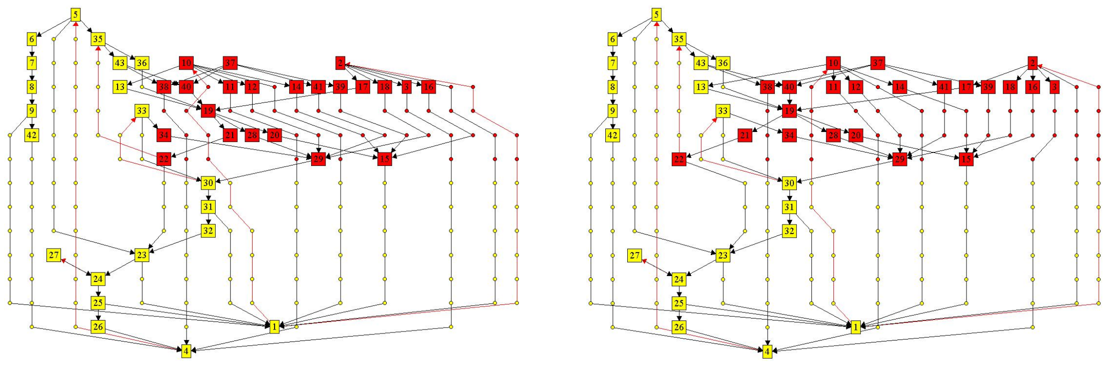


Figure 6.5: Improvement of a selected graph drawing region (of the Forrester's World Dynamics graph.)

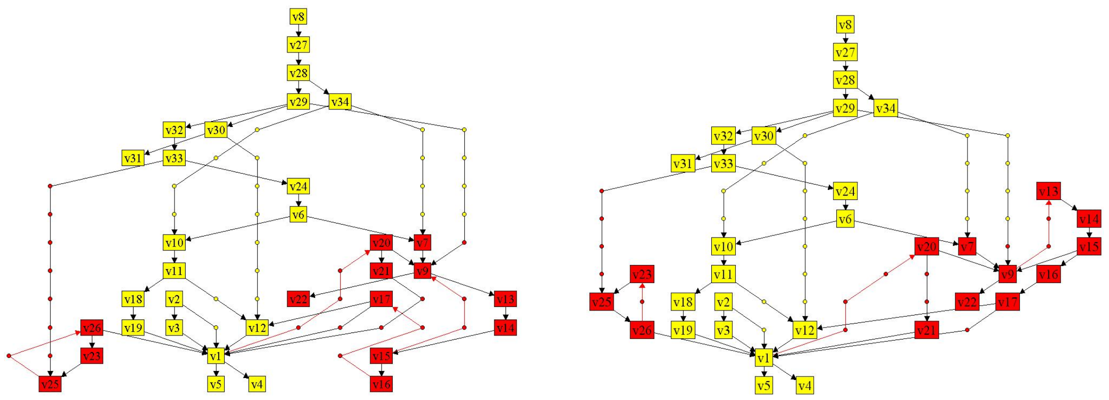


Figure 6.6: Simultaneous improvement of two disjoint selected graph drawing regions (of the C-Language Syntax graph.)

6.5 Evaluation

We performed human experiments, similarly to what was done in the previous chapter, in order to verify whether the combination of user plus system could provide better drawings than the genetic algorithm running alone.

6.5.1 Experiment Setup

The experiments involved 20 subjects, all students from the School of Information Technologies of The University of Sydney. Seven subjects were doing an honors degree, eight were doing masters, and five were Ph.D. students. They had different interests in Computer Science; some students were involved with research in Language Technologies and Networking. Others were doing a coursework masters degree in Information Technologies in general. Two honors and two Ph.D students were from the Information Visualization Research Group. None of the subjects, however, had been involved with the User Hints research before.

A 30-minute introduction to the system was provided. During the introduction the subjects learned about the graph drawing problem, the tools available in the system, and the goal of the experiments. They then had 15 minutes for practicing with the system, by improving a drawing of the Ecosystem graph (this is the graph *G3* used in the Pilot Study in Section 5.5). The drawings produced for the Ecosystem were not included in the analysis of the performance of the subjects.

Each subject did 12 experiments[2], divided in two task sessions as shown in Table 6.1. The experiments used graphs mostly from graph drawing papers. The graphs *Csyntax*, *Klayer*, *Unixsys*

[2]The experiments were performed in a standard computer, a PC Intel Celeron 702Mhz with 256Mb RAM, running MS Windows 98.

Exp.	Graph Name	|V|	|E|	Initial Drawing	User Interaction
Session 1					
A1	Csyntax	34	46	Sugiyama	All
A2	Klayer	18	24	Sugiyama	All
A3	Unixsys	41	49	Sugiyama	All
A4	Worlddyn	43	69	Sugiyama	All
A5	Knation	23	87	Sugiyama	All
A6	Telcall	111	193	Sugiyama	All
A7	Gd94dir	40	131	Sugiyama	All
Session 2					
B1	Knation	23	87	Sugiyama	Focus Only
B2	Telcall	111	193	Sugiyama	Focus Only
B3	Unixsys (with constraints)	41	49	Sugiyama	All
B4	Worlddyn	43	69	High Quality	All
B5	Gd94dir	40	131	High Quality	All

Table 6.1: Setup for the experiments with the genetic algorithm.

and *Worlddyn* are the same presented in Chapter 5. The *Knation* is the Knowledge Nation graph presented in [26]. The graph *Telcall* is a network of telephone calls. It appeared in the Graph Drawing Contest during the Graph Drawing Conference in 1996 [60]. Another graph from the Graph Drawing Contest, this time from 1994 [59] is the *Gd94dir*; it describes the references between graph drawing papers. The size of all graphs are shown in the table. The graphs *Worlddyn*, *Knation*, *Telcall* and *Gd94dir* are the largest ones.

The first task session consisted of 7 experiments (labeled *A1* to *A7*), where the subjects had to improve drawings created by the Sugiyama method. The subjects could perform manual changes of the drawings and focus the genetic algorithm for improving the layout of a group of vertices.

The second task session involved 5 experiments (*B1* to *B5*), where the subjects had to execute special graph drawing activities. In experiments *B1* and *B2* the subjects could perform only focus; manual changes were disabled in this setup. In experiment *B3* the subjects could perform both focus and manual changes, but the graph had pre-assigned 16 layout constraints (9 Left-Right and 7 Top-Down constraints) that needed to be satisfied. In experiments *B4* and *B5* the subjects were presented with high quality initial drawings, previously computed by running the genetic algorithm extensively on the Sugiyama-based drawings. The task was to try to improve the high quality by performing focus and manual changes.

We allowed 20 minutes for every experiment. The order of the drawing activities for the first and the second sessions was sorted randomly, so that no two subjects did the same experiment in the same order. The only exception was the experiment *A1*, which was the first task for all subjects.

The experiments were video taped and the tapes were studied afterward. We also observed the subjects during the experiments in a non-invasive way. We noted any significant problem with the interactive framework and promising graph drawing strategies. At the end of the experiments, the subjects were interviewed about the strong and weak points of the system, their general strategy for using the genetic algorithm, and about the amount of time available for the graph drawing tasks.

In order to compare the human performance with a fully automatic approach, we produced drawings by running the genetic algorithm alone: for every experiment[3] we ran 100 genetic algorithms, each one taking 40 minutes. The algorithms were executed on the same type of computer used by the subjects, and the drawings they produced after intervals of 20 and 40 minutes were collected. The results for 20 minutes indicate what the genetic algorithm can produce if they are used instead of the humans during the experiments. The results for 40 minutes show the performance of the algorithm if extra 20 minutes are allowed.

6.5.2 Results

The results of the experiments are presented in Tables 6.2 to 6.13. Each table is divided into four parts that show, from top to bottom, the quality of the initial drawing, the quality of the drawings produced by the subjects, and a summary of the results obtained by the genetic algorithm for 20 and 40 minutes (*GA20* and *GA40* respectively).

The table has two extra columns for the subjects indicating their performance against the fully-automatic genetic algorithm: columns *GA20Comp* and *GA40Comp*. These columns contain the number of executions of the genetic algorithm (for 20 and 40 minutes respectively) that produced worse results than a particular human subject. We call this information the *human performance*. It is important to have such measure since the genetic algorithm is a stochastic method (it may produce different results in every execution), and we need to know about how the subjects perform when compared to possible outputs of the method. A value 100 in *GA20Comp* (or *GA40Comp*) indicates that the subject in the corresponding row performed better than all 100 executions of the genetic algorithm. A value 0, on the other end, means that the subject performed worse than all executions of the algorithm, or as bad as its worst execution. In general, a value greater than 50 indicates that the human performance was higher than the average performance of the genetic algorithm. For summarizing the human results, we provide the minimum, the maximum, the average and the

[3]Except experiments *B1* and *B2*, which shared the automatic-generated results for experiments *A5* and *A6*, respectively.

standard deviation for every column related to the subjects[4].

For the genetic algorithm, the tables present the quality of the best drawing, the fiftieth best drawing (row *Best50*), and of the worst drawing produced by the 100 executions for 20 and 40 minutes. The minimum, the maximum, the average and the standard deviation values for each aesthetic criterion separately are also included.

The first good result of the evaluation was that the genetic algorithm, by itself, could improve the initial drawings created by the Sugiyama method significantly. The genetic algorithm was even better than the human subjects in the previous study (in Chapter 5) for two graphs: *Csyntax* and *Worlddyn*. For the *Worlddyn* graph, for example, the execution of the genetic algorithm resulted in a drawing with only 21 edge crossings (see Table 6.2), while the best human performance was 35 crossings (Table 5.2); even the 50th best execution of the genetic algorithm produced a better result: 33 edge crossings. On the other hand, the human results in the pilot study were still better for graphs *Klayer* and *Unixsys*.

We highlight on the tables the cases where the human performance was 100, in order to help the reader to locate those values. We also highlight the *average human performance*, which are given by the intersections of row *Av* with columns *GA20Comp* and *GA40Comp*.

In experiment *A1* (graph *Csyntax*), the subjects were more effective than the genetic algorithm alone: 6 out of 20 subjects outperformed the algorithm in all executions. Note that some subjects did very bad (Subjects 2, 10, 17, 13 and 11), with a performance inferior to 40. Nevertheless, the average human performance was reasonably high: 68.05.

For the experiments *A2* (graph *Klayer*) and *A3* (graph *Unixsys*) the human performance was outstanding: 18 and 15 subjects, respectively, had performance 100. The average human performance was above 90, and just one subject – in experiment *A3* – had a very poor performance.

We found that longer executions of the genetic algorithm for graphs *Csyntax*, *Klayer* and *Unixsys* could not improve the drawings much. The results for *GA40* suggest that the genetic algorithm had reached a local minimum in the first 20 minutes, and could not escape from it.

For the largest graphs in the first experiment session, the human performance (*GA20Comp*) was low. In experiment *A4* (with graph *Worlddyn*) only one subject had performance 100; the average human performance was 64.60 – still higher than the average execution of the genetic algorithm. In experiments *A5*, *A6* and *A7* (graphs *Knation*, *Telcall* and *Gd94dir*), none of the subjects achieved

[4]Note that a lower value of the latter quality aesthetic does not mean necessarily a better drawing, since there is a priority order for the aesthetic criteria. As we mentioned in Chapter 5, less offending edges and less edge crossings may imply in a greater number of dummy vertices and bends, or in a larger drawing area.

performance 100, and the average human performance was 38.80, 32.00 and 29.29, respectively. Note that the genetic algorithm produced better drawings when more time was allowed for these graphs (see rows for *G40* and column *GA40Comp* for more details).

The comments made by the subjects during the interview help to understand their performance in the first session of experiments: the subjects said that they could use imagination for manually improving the drawings of the small graphs. For large graphs they mentioned feeling lost with the overwhelming number of edge crossings and bends.

In fact the aesthetic measures of the initial drawings of the large graphs were very bad, and much work would be necessary to produce better solutions. Running the genetic algorithm on the whole graph for these cases would be faster than by manual change. Most of the subjects realized this fact, but too late in the experiments. For instance, we saw some subjects spending time with local edge crossings in the initial drawing of *Telcall*, which we knew could be eliminated by running the genetic algorithm to improve the drawing globally. Note that the graph *Telcall* is a bipartite, and the Sugiyama method emphasizes this characteristic by producing a drawing using two layers (see Figure 6.13(a)). The minimum number of edge crossings is obtained by expanding the drawing to use more layers.

In the second session of the experiments the subjects had to deal with different situations, which included using focus only, manipulating layout constraints (with higher priority than the normal aesthetic criteria), and working on initial drawings with high quality.

In experiment *B1* and *B2* (graphs *Knation* and *Telcall*, respectively), we expected the subjects to achieve very good results. The motivation for these experiments was to avoid having the system spend processing time on areas of the drawing which were already of good quality. This is illustrated in Figure 6.7, where the layout of the edges on the left-hand side of the drawing can be improved. Focusing the genetic algorithm on those edges produces the intended layout must faster than running it on the whole graph. The subjects did use this approach, by refocusing the genetic algorithm on a new part of the drawing after 12 to 24 seconds without significant improvement. Nevertheless, the overall results were not good for two reasons:

1. Some subjects performed focus locally at first; however, the initial drawing could be improved more by running the genetic algorithm on the entire graph.

2. The genetic algorithm reached local minima, and the subjects could not use any other tool but focus to help it to escape from the minima.

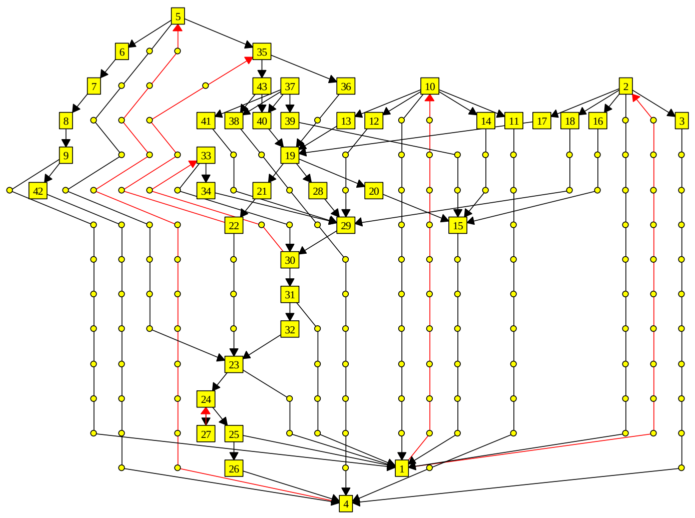


Figure 6.7: Drawing of the *Worlddy* graph with a promising area for improvement, on the left-hand side of the figure.

The average human performance in experiments *B1* and *B2* is a little better than the results in the corresponding experiments in the first session (*A5* and *A6*). For example, in *B1* the average human performance *GA20Comp* is 45.60, while in *A5* it was 38.80. This difference seems to be related to the fact that using the genetic algorithm for large graphs is better than performing improvements via manual changes. The experience gained from the experiments in the first session may also have contributed to a better human performance.

In experiment *B3*, the system started with the same initial drawing used in experiment *A3*, but added 16 layout constraints to it. Constraints were considered more important than the normal aesthetic criteria. Twelve constraints were unsatisfied in the initial solution. One subject had performance 100, and the average human performance was 86.95. The genetic algorithm was able to help the subjects in solving almost all unsatisfied constraints in a first run on the whole graph. However, it had difficulty with left-right constraints, and very often got stuck in a local minimum where one such constraint was unsatisfied. The subjects could solve this problem by simply reordering the vertices on the layers and running the algorithm again to obtain further improvements. Note that the genetic algorithm executing alone for 20 minutes in a constrained problem was not able to escape from local minima. The average and the *Best50* results of the genetic algorithm are worse than the subject results not only in terms of constraint satisfaction, but also considering offending edges and edge crossings. Even allowing 20 more minutes to the algorithm (in *GA40*) did not improve the drawings.

Finally, in experiments *B4* and *B5*, with high quality initial drawings, we got surprising results.

In experiment *B4* (with graph *Worlddyn*), nine subjects had performance 100. The average human performance was 70.80 for *GA20Comp*, and 69.70 for *GA40Comp*. This is much better than the results in experiment *A4*, where the average human performances were 64.60 and 63.90, respectively, with just one subject having performance 100. Such improvement was obtained because of the use of manual changes. The genetic algorithm by itself could not produce the same type of results, as we demonstrate in rows *GA20* and *GA40* of Table 6.12.

In experiment *B5* (graph *Gd94dir*), seven subjects had performance 100 for *GA20Comp*, and the average human performance was 89.10. For *GA40Comp*, four subjects had performance 100, and the average was 83.55, which indicates that longer executions of the genetic algorithm could improve the automatic results. Nevertheless, the subjects performed much better than in experiment *A7*, where the average human performances were only 29.70 (*GA20Comp*) and 24.95 (*GA40Comp*), and no subject had performance 100.

The results in experiments *B4* and *B5* show that human intervention can help when the genetic algorithm becomes stuck in a local minimum or is already close to it. The results also show that not all subjects could help the system. In experiment *B4*, six subjects had poor performance – *GA20Comp* was lower than 10.

Table 6.14 shows the overall relationship between the users' actions and the performance of the subjects (this includes the sum of parameters from all experiments, except from *B1* and *B2* – these two experiments had no manual changes, and, therefore, its inclusion would incorrectly affect the correlation analysis shown in the table for manual moves). The table contains the level of study, *expected knowledge* on Graph Drawing, sex, information regarding the execution of the genetic algorithm and manual changes of the drawings, and the *total performance* for every subject. The *expected drawing knowledge* is a number between 1 and 3 that we assigned to the subjects based on our knowledge about their familiarity with the Graph Drawing area. Value 3 was given to subjects who have already worked on Graph Drawing; this includes some of the members from the Information Visualization Research group or from other research groups. Value 2 was given to other PhD students. All other subjects received value 1. The column *GA Time* gives the sum of the processing time (in seconds) of the genetic algorithm activated by the subjects during the experiments. The column *GA Exec* is the number of times that the genetic algorithm was executed. The column *Interactive Time* provides an estimation of how much time (in seconds) the subjects spent with manual changes of the drawings. This parameter consists of the sum of all intervals of time of 25 seconds during which the user performed at least one action related to manual change and did not execute the genetic algorithm. The column *Total Performance* for a subject gives the sum of his/her *GA20Comp* performance plus the number of other subjects who performed worse than him/her in every experiment. The correlations of the main columns of the table with the column *Total Performance* is presented. The information about the subjects is sorted by increasing order of *total performance*.

Note that the *level of study*, the *expected knowledge* and the sex of the subjects did not affect their overall performance significantly. We noticed, however, that there was a higher positive correlation between the use of the genetic algorithm and the human performance. We also found a negative correlation between the time spent with manual changes of the drawings and the human performance[5]. Even though there are peculiarities in how this correlation is observed for the individual experiments, the overall conclusion is that using the genetic algorithm provides better results than performing many manual changes. We know, however, that human intervention is useful since the subjects could not outperform the genetic algorithm in several cases without such a tool. The question is when and how the user should interact with the optimization method. The experiments with small graphs and with high initial drawings suggests the answer: interaction is useful when the genetic algorithm becomes stuck in local minima in order to help it to progress towards a better solution.

By observing the subjects doing the experiments we identified several graph drawing configurations that represent local minima to the genetic algorithm. Some of the minima represent difficulty in:

- Moving a subgraph consisting of many vertices to a different position – this consisted of a local minimum when the vertices had to be moved as whole group, otherwise a worse drawing would be created.
- Eliminating unnecessary columns and layers containing only dummy vertices or no vertices at all in the middle of the drawing so that the used area is reduced – the genetic algorithm could solve such a problem more easily when the unnecessary columns and layers were at the extreme ends (left, right, top or bottom) of the drawing.
- Reducing edge crossings by routing the edges in a non-straight path – if the routing involves moving dummy vertices to horizontal positions distant from the endpoints of the edges then

[5]These two correlations are expected to be opposite, as the time spent using the genetic algorithm and the time used for manual changes are in some sense complementary.

the algorithm may not be able to find this solution.

In those configurations the subjects improved the drawing by doing the necessary change by hand. Nevertheless, they also developed some interesting general strategies for escaping from local minima:

- Creating empty columns and layers in the center of the drawing that can be used later by the genetic algorithm as a temporary space to reorganize the vertices (note that this is exactly the opposite of what we mentioned before, but it helps the genetic algorithm to solve edge crossings in some cases).
- Swapping the position of the dummy vertices of two edges that are crossing.
- Attempting to escape from local minima by ”messing up” the drawing, that is, moving several vertices randomly and then running the genetic algorithm to create a new improved layout; if the new layout is not better than the previous one, then the best existing solution can be recovered.

The initial drawings of the graphs and some improved drawings created during the experiments are illustrated in Figures 6.8 to 6.19.

The final conclusions of the experiments are that:

- The subjects could help the genetic algorithm to escape from local minima.
- The general strategy defined by the subjects for improving the drawings was to run the genetic algorithm on the whole graph until no further significant improvement could be obtained. Then they would alternate manual changes with focused re-executions of the algorithm several times. Manual changes were necessary to escape from local minima. Re-executing the genetic algorithm after a manual change would improve the quality of “draft solutions” created by the subjects (that is, performing a fine tuning of the drawing) and progress the search towards a better solution.
- The final quality of the drawings in the experiments depended on human skills; while some subjects did very well, others performed poorly. We could not clearly identify what type of skills makes the difference. However, it seems that geometrical thinking is one of the capabilities necessary for this kind of task.

Results for Csyntax		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		4	12	39	19	182		
Subjects	1	3	5	74	14	260	89	89
	2	4	11	39	12	182	0	0
	3	3	6	46	5	187	58	58
	4	3	6	35	5	195	61	61
	5	3	3	63	8	247	100	100
	6	3	3	45	6	255	100	100
	7	3	6	50	3	216	55	55
	8	3	6	30	3	182	87	87
	9	3	4	56	5	323	99	99
	10	3	9	34	7	210	1	1
	11	3	6	57	11	198	36	36
	12	3	3	72	12	260	100	100
	13	3	6	73	8	294	22	22
	14	3	3	68	27	342	100	100
	15	3	6	46	5	208	57	57
	16	3	3	51	4	198	100	100
	17	3	7	72	17	285	12	12
	18	3	5	30	5	154	98	98
	19	3	6	30	3	196	86	86
	20	3	3	51	5	234	100	100
	Min	3	3	30	3	154	0	0
	Max	4	11	74	27	342	100	100
	Av	3.05	5.35	51.10	8.25	231.30	68.05	68.05
	StD	0.22	2.13	15.37	5.93	50.65	36.24	36.24
GA20	Best	3	4	55	6	234		
	50Best	3	6	54	5	270		
	Worst	3	9	34	8	224		
	Min	3	4	30	3	140		
	Max	3	9	80	9	336		
	Av	3.00	6.18	46.20	5.77	211.51		
	Std	0.00	0.87	15.53	1.48	44.18		
GA40	Best	3	4	55	6	234		
	50Best	3	6	54	5	270		
	Worst	3	9	34	8	224		
	Min	3	4	30	3	140		
	Max	3	9	80	9	336		
	Av	3.00	6.18	46.20	5.76	211.51		
	StD	0.00	0.87	15.53	1.46	44.18		

Table 6.2: Results of the experiment *A1*, for the graph *Csyntax*.

Results for Klayer		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	5	0	0	24		
Subjects	1	0	2	4	1	35	100	100
	2	0	1	14	3	56	100	100
	3	0	2	11	2	56	99	99
	4	0	1	11	2	49	100	100
	5	0	1	10	2	48	100	100
	6	0	1	11	0	119	100	100
	7	0	2	4	1	35	100	100
	8	0	1	11	1	70	100	100
	9	0	1	11	2	63	100	100
	10	0	2	4	0	50	100	100
	11	0	1	11	1	63	100	100
	12	0	1	11	3	70	100	100
	13	0	1	9	1	48	100	100
	14	0	1	9	3	42	100	100
	15	0	1	15	4	72	100	100
	16	0	1	9	1	48	100	100
	17	0	1	13	5	63	100	100
	18	0	2	13	4	56	99	99
	19	0	2	4	1	35	100	100
	20	0	1	11	3	56	100	100
	Min	0	1	4	0	35	99	99
	Max	0	2	15	5	119	100	100
	Av	0.00	1.30	9.80	2.00	56.70	99.90	99.90
	StD	0.00	0.47	3.35	1.38	18.57	0.31	0.31
GA20	Best	0	2	4	1	40		
	50Best	0	4	3	0	35		
	Worst	0	4	3	1	30		
	Min	0	2	3	0	25		
	Max	0	4	8	2	54		
	Av	0.00	3.85	3.33	0.16	35.87		
	Std	0.00	0.39	0.92	0.39	4.17		
GA40	Best	0	2	4	1	40		
	50Best	0	4	3	0	35		
	Worst	0	4	3	1	30		
	Min	0	2	3	0	25		
	Max	0	4	8	2	54		
	Av	0.00	3.85	3.33	0.16	35.87		
	StD	0.00	0.39	0.92	0.39	4.17		

Table 6.3: Results of the experiment *A2*, for the graph *Klayer*.

Results for Unixsys		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	4	24	9	132		
Subjects	1	0	2	22	5	143	98	98
	2	0	3	24	5	143	2	2
	3	0	0	32	4	156	100	100
	4	0	0	30	2	168	100	100
	5	0	0	25	4	154	100	100
	6	0	0	25	2	187	100	100
	7	0	0	25	3	143	100	100
	8	0	1	43	4	210	98	98
	9	0	0	25	3	154	100	100
	10	0	0	25	4	44	100	100
	11	0	0	25	7	143	100	100
	12	0	0	25	4	121	100	100
	13	0	0	49	5	288	100	100
	14	0	0	25	5	154	100	100
	15	0	0	28	2	154	100	100
	16	0	0	25	3	154	100	100
	17	0	3	22	2	132	97	97
	18	0	1	30	4	168	98	98
	19	0	0	25	3	154	100	100
	20	0	0	28	3	132	100	100
	Min	0	0	22	2	44	2	2
	Max	0	3	49	7	288	100	100
	Av	0.00	0.50	27.90	3.70	155.10	94.65	94.65
	StD	0.00	1.00	6.76	1.30	44.24	21.83	21.83
GA20	Best	0	1	27	4	143		
	50Best	0	3	24	4	132		
	Worst	0	3	24	6	132		
	Min	0	1	24	2	121		
	Max	0	3	27	7	154		
	Av	0.00	2.95	24.06	3.93	135.96		
	Std	0.00	0.30	0.42	0.88	7.58		
GA40	Best	0	1	27	4	143		
	50Best	0	3	24	4	132		
	Worst	0	3	24	6	132		
	Min	0	1	24	2	121		
	Max	0	3	27	7	154		
	Av	0.00	2.95	24.06	3.93	135.96		
	StD	0.00	0.30	0.42	0.88	7.58		

Table 6.4: Results of the experiment *A3*, for the graph *Unixsys*.

Results for Worlddyn		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		6	70	144	56	360		
Subjects	1	6	33	148	26	400	59	56
	2	6	31	235	40	500	67	67
	3	6	34	172	24	425	39	38
	4	6	22	202	33	408	98	98
	5	6	23	229	37	588	97	97
	6	6	26	184	41	456	90	90
	7	6	28	145	33	360	88	88
	8	6	31	148	26	345	78	77
	9	6	18	164	18	384	100	100
	10	6	63	199	64	456	0	0
	11	6	57	136	21	360	0	0
	12	6	25	456	59	1026	92	92
	13	6	24	172	33	391	95	95
	14	6	70	144	56	360	0	0
	15	6	28	193	28	432	87	87
	16	6	33	177	28	368	50	46
	17	6	38	182	46	442	4	4
	18	6	21	203	32	396	99	99
	19	6	33	169	41	396	50	46
	20	6	22	164	29	368	99	98
	Min	6	18	136	18	345	0	0
	Max	6	70	456	64	1026	100	100
	Av	6.00	33.00	191.10	35.75	443.05	64.60	63.90
	StD	0.00	14.21	68.05	12.55	148.70	37.45	37.63
GA20	Best	6	21	178	24	456		
	50Best	6	33	155	26	384		
	Worst	6	43	154	28	400		
	Min	6	21	139	21	360		
	Max	6	43	196	36	480		
	Av	6.00	33.06	154.87	25.38	395.24		
	Std	0.00	4.37	9.59	2.87	29.38		
GA40	Best	6	21	178	24	456		
	50Best	6	33	153	23	400		
	Worst	6	43	154	28	400		
	Min	6	21	139	21	360		
	Max	6	43	185	35	475		
	Av	6.00	32.96	154.21	25.20	393.32		
	StD	0.00	4.35	8.23	2.66	26.47		

Table 6.5: Results of the experiment *A4*, for the graph *Worlddyn*.

Results for Knation		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		30	249	186	137	460		
Subjects	1	28	135	216	78	546	35	32
	2	28	166	274	91	555	3	2
	3	28	117	267	82	765	74	73
	4	28	140	236	79	481	30	24
	5	28	127	295	112	608	57	54
	6	28	101	242	86	546	93	93
	7	28	132	180	82	407	43	37
	8	28	147	244	77	645	18	13
	9	28	82	166	68	385	99	99
	10	28	156	134	59	396	9	5
	11	28	136	270	109	585	34	29
	12	28	123	241	119	812	66	61
	13	28	136	180	81	530	35	32
	14	29	509	221	172	720	0	0
	15	28	143	192	65	564	26	19
	16	28	117	204	77	432	74	73
	17	30	204	202	121	552	0	0
	18	28	140	547	126	1175	29	23
	19	29	96	203	103	564	0	0
	20	28	130	230	84	533	51	44
	Min	28	82	134	59	385	0	0
	Max	30	509	547	172	1175	99	99
	Av	28.20	151.85	237.20	93.55	590.05	38.80	35.65
	StD	0.52	87.96	83.13	26.67	178.76	30.60	30.94
GA20	Best	28	78	288	82	675		
	50Best	28	130	235	88	598		
	Worst	28	190	264	94	765		
	Min	28	78	126	60	378		
	Max	28	190	707	112	2016		
	Av	28.00	130.01	263.60	85.21	681.14		
	Std	0.00	19.88	69.74	10.18	190.24		
GA40	Best	28	78	288	82	675		
	50Best	28	128	496	93	1880		
	Worst	28	178	266	94	700		
	Min	28	78	126	60	378		
	Max	28	178	496	103	1880		
	Av	28.00	127.22	256.15	82.51	667.39		
	StD	0.00	18.13	56.64	9.58	173.06		

Table 6.6: Results of the experiment *A5*, for the graph *Knation*.

Results for Telcall		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	1740	0	0	192		
Subjects	1	0	380	403	172	2074	68	59
	2	0	431	290	138	1495	37	25
	3	0	435	635	126	2140	35	25
	4	0	361	213	117	1177	84	71
	5	0	628	530	239	2508	2	0
	6	0	442	310	144	2366	33	25
	7	0	388	403	120	2014	63	51
	8	0	309	224	129	1440	99	96
	9	0	382	245	176	1190	68	57
	10	0	581	124	82	728	5	2
	11	0	966	198	169	1166	0	0
	12	0	622	303	188	2567	3	0
	13	0	653	411	279	1272	1	0
	14	0	658	493	379	1875	1	0
	15	0	514	502	236	2489	11	8
	16	0	607	215	132	1053	4	1
	17	0	1122	47	47	288	0	0
	18	0	434	374	133	1392	35	25
	19	0	356	227	129	1296	87	78
	20	0	594	416	206	1728	4	1
	Min	0	309	47	47	288	0	0
	Max	0	1122	635	379	2567	99	96
	Av	0.00	543.15	328.15	167.05	1612.90	32.00	26.20
	StD	0.00	206.17	147.67	74.04	628.67	34.23	31.22
GA20	Best	0	280	252	109	1540		
	50Best	0	407	620	134	3304		
	Worst	0	664	555	168	2352		
	Min	0	280	156	87	981		
	Max	0	664	1079	206	3848		
	Av	0.00	425.52	360.03	128.64	1785.96		
	Std	0.00	74.79	164.77	26.78	578.18		
GA40	Best	0	277	233	97	1540		
	50Best	0	388	515	144	2484		
	Worst	0	622	147	83	880		
	Min	0	277	147	82	880		
	Max	0	622	999	193	3416		
	Av	0.00	401.87	330.30	115.68	1695.48		
	StD	0.00	66.74	158.65	22.93	545.36		

Table 6.7: Results of the experiment *A6*, for the graph *Telcall*.

Results for Gd94dir		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	757	325	203	742		
Subjects	1	0	494	362	143	864	93	84
	2	0	721	325	174	742	0	0
	3	0	525	372	133	952	20	15
	4	0	521	352	130	864	37	23
	5	0	515	381	138	884	55	42
	6	0	495	684	180	2160	90	82
	7	0	523	380	148	864	28	17
	8	0	536	359	155	952	4	2
	9	0	534	339	134	870	6	4
	10	0	562	335	157	780	0	0
	11	0	576	335	156	795	0	0
	12	0	686	682	354	1426	0	0
	13	0	540	585	191	1988	1	0
	14	0	497	480	141	1100	89	80
	15	0	515	646	193	1200	55	40
	16	0	542	443	181	972	0	0
	17	0	698	323	180	742	0	0
	18	0	525	412	162	880	19	15
	19	0	614	333	181	795	0	0
	20	0	479	393	154	880	97	95
	Min	0	479	323	130	742	0	0
	Max	0	721	684	354	2160	97	95
	Av	0.00	554.90	426.05	169.25	1035.50	29.70	24.95
	StD	0.00	70.33	122.47	47.86	391.60	36.60	33.57
GA20	Best	0	458	416	160	901		
	50Best	0	517	420	135	936		
	Worst	0	542	370	123	986		
	Min	0	458	335	113	765		
	Max	0	542	615	162	1870		
	Av	0.00	514.03	383.86	132.16	910.56		
	Std	0.00	15.39	44.68	11.10	126.44		
GA40	Best	0	450	416	168	816		
	50Best	0	512	340	117	840		
	Worst	0	540	359	129	848		
	Min	0	450	335	115	765		
	Max	0	540	500	168	1265		
	Av	0.00	509.16	381.23	128.32	897.16		
	StD	0.00	16.94	33.74	9.72	82.24		

Table 6.8: Results of the experiment *A7*, for the graph *Gd94dir*.

Results for Knation Focus		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		30	249	186	137	460		
Subjects	1	28	144	264	92	675	26	19
	2	28	122	287	86	690	67	61
	3	28	112	238	87	602	80	79
	4	28	124	242	75	616	65	60
	5	28	129	299	88	672	54	48
	6	28	120	218	83	559	71	67
	7	28	172	200	83	494	3	2
	8	28	126	251	85	750	58	56
	9	28	144	221	83	616	26	19
	10	28	132	233	83	533	43	36
	11	28	118	258	90	585	73	70
	12	28	143	238	81	658	26	19
	13	28	136	180	81	530	35	32
	14	28	132	240	79	624	43	36
	15	28	129	266	78	795	55	49
	16	28	126	254	87	672	58	56
	17	28	157	192	84	624	9	4
	18	28	154	247	80	672	11	6
	19	28	147	213	73	624	18	13
	20	28	105	239	78	616	91	90
	Min	28	105	180	73	494	3	2
	Max	28	172	299	92	795	91	90
	Av	28.00	133.60	239.00	82.80	630.35	45.60	41.10
	StD	0.00	16.27	29.69	4.85	72.47	25.48	26.33
GA20	Best	28	78	288	82	675		
	50Best	28	130	235	88	598		
	Worst	28	190	264	94	765		
	Min	28	78	126	60	378		
	Max	28	190	707	112	2016		
	Av	28.00	130.01	263.60	85.21	681.14		
	Std	0.00	19.88	69.74	10.18	190.24		
GA40	Best	28	78	288	82	675		
	50Best	28	128	496	93	1880		
	Worst	28	178	266	94	700		
	Min	28	78	126	60	378		
	Max	28	178	496	103	1880		
	Av	28.00	127.22	256.15	82.51	667.39		
	StD	0.00	18.13	56.64	9.58	173.06		

Table 6.9: Results of the experiment *B1*, for the graph *Knation* with the subjects performing only focus.

Results for Telcall Focus		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	1740	0	0	192		
Subjects	1	0	371	223	121	1554	79	66
	2	0	556	339	155	1800	6	3
	3	0	404	679	135	3596	51	35
	4	0	356	225	114	1210	87	78
	5	0	462	169	107	952	31	19
	6	0	486	147	92	928	20	12
	7	0	518	400	188	1443	10	7
	8	0	474	249	117	1526	23	16
	9	0	379	271	117	1534	70	59
	10	0	390	158	98	1122	62	47
	11	0	420	157	84	832	42	28
	12	0	432	196	113	1078	37	25
	13	0	605	283	132	1744	4	1
	14	0	530	181	114	1155	9	5
	15	0	495	212	119	1050	16	10
	16	0	342	212	90	1391	94	87
	17	0	551	279	132	1017	7	3
	18	0	376	338	142	1887	73	60
	19	0	651	210	95	900	1	0
	20	0	379	308	126	1272	70	59
	Min	0	342	147	84	832	1	0
	Max	0	651	679	188	3596	94	87
	Av	0.00	458.85	261.80	119.55	1399.55	39.60	31.00
	StD	0.00	88.44	119.86	24.36	605.07	31.14	28.24
GA20	Best	0	280	252	109	1540		
	50Best	0	407	620	134	3304		
	Worst	0	664	555	168	2352		
	Min	0	280	156	87	981		
	Max	0	664	1079	206	3848		
	Av	0.00	425.52	360.03	128.64	1785.96		
	Std	0.00	74.79	164.77	26.78	578.18		
GA40	Best	0	277	233	97	1540		
	50Best	0	388	515	144	2484		
	Worst	0	622	147	83	880		
	Min	0	277	147	82	880		
	Max	0	622	999	193	3416		
	Av	0.00	401.87	330.30	115.68	1695.48		
	StD	0.00	66.74	158.65	22.93	545.36		

Table 6.10: Results of the experiment *B2*, for the graph *Telcall* with the subjects performing only focus.

Results for Unixsys_Constr		Constr.	Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		12	0	4	24	9	132		
Subjects	1	0	4	10	44	12	391	80	80
	2	0	3	4	50	8	416	94	94
	3	0	3	2	50	8	352	99	99
	4	0	3	3	72	19	462	96	96
	5	0	3	3	70	23	468	96	96
	6	0	3	8	46	10	375	87	87
	7	0	3	2	37	9	273	100	100
	8	0	4	6	44	8	294	83	83
	9	0	3	4	38	12	255	95	95
	10	0	3	7	59	18	384	88	88
	11	1	4	4	37	8	253	46	46
	12	0	3	6	54	16	357	89	89
	13	0	3	9	55	11	540	86	86
	14	0	3	5	74	19	418	90	90
	15	0	3	6	122	23	576	88	88
	16	0	3	6	47	9	368	89	89
	17	0	3	5	98	33	437	90	90
	18	1	3	6	89	20	272	61	61
	19	0	3	2	63	15	270	99	99
	20	0	4	5	52	18	570	83	83
	Min	0	3	2	37	8	253	46	46
	Max	1	4	10	122	33	576	100	100
	Av	0.10	3.20	5.15	60.05	14.95	386.55	86.95	86.95
	StD	0.31	0.41	2.25	22.15	6.71	101.49	12.98	12.98
GA20	Best	0	3	2	42	7	336		
	50Best	1	3	10	81	14	306		
	Worst	2	7	7	22	5	208		
	Min	0	3	0	19	2	176		
	Max	2	7	13	95	25	667		
	Av	0.77	4.06	6.00	52.05	10.86	303.02		
	Std	0.51	1.09	2.95	16.14	3.70	82.82		
GA40	Best	0	3	2	42	7	336		
	50Best	1	3	10	81	14	306		
	Worst	2	7	7	22	5	208		
	Min	0	3	0	19	2	176		
	Max	2	7	13	95	25	667		
	Av	0.77	4.06	6.00	52.03	10.86	303.02		
	StD	0.51	1.09	2.95	16.14	3.70	82.82		

Table 6.11: Results of the experiment *B3*, for the graph *Unixsys* with layout constraints.

Results for Worlddyn HQ		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		6	22	152	24	375		
Subjects	1	6	22	152	24	360	0	0
	2	6	20	221	36	500	6	1
	3	6	20	154	29	375	95	95
	4	6	17	214	28	437	100	100
	5	6	15	242	41	528	100	100
	6	6	19	178	31	450	99	99
	7	6	21	150	22	384	6	1
	8	6	20	147	25	345	98	98
	9	6	17	176	24	391	100	100
	10	6	22	145	21	360	5	1
	11	6	22	152	23	360	5	1
	12	6	16	167	28	408	100	100
	13	6	19	235	29	475	99	99
	14	6	17	189	40	425	100	100
	15	6	16	165	27	368	100	100
	16	6	14	185	26	408	100	100
	17	6	20	147	22	375	98	98
	18	6	21	226	31	456	5	1
	19	6	16	189	33	408	100	100
	20	6	17	181	30	425	100	100
	Min	6	14	145	21	345	0	0
	Max	6	22	242	41	528	100	100
	Av	6.00	18.55	180.75	28.50	411.90	70.80	69.70
	StD	0.00	2.50	31.59	5.68	50.29	44.56	46.27
GA20	Best	6	18	158	27	400		
	50Best	6	20	155	26	375		
	Worst	6	22	152	23	375		
	Min	6	18	147	22	375		
	Max	6	22	158	30	400		
	Av	6.00	20.09	155.45	25.70	381.50		
	Std	0.00	0.49	1.78	1.70	11.02		
GA40	Best	6	18	158	27	400		
	50Best	6	20	155	25	375		
	Worst	6	22	152	23	375		
	Min	6	18	147	22	375		
	Max	6	22	158	30	400		
	Av	6.00	20.00	155.57	25.76	381.50		
	StD	0.00	0.28	1.63	1.60	11.02		

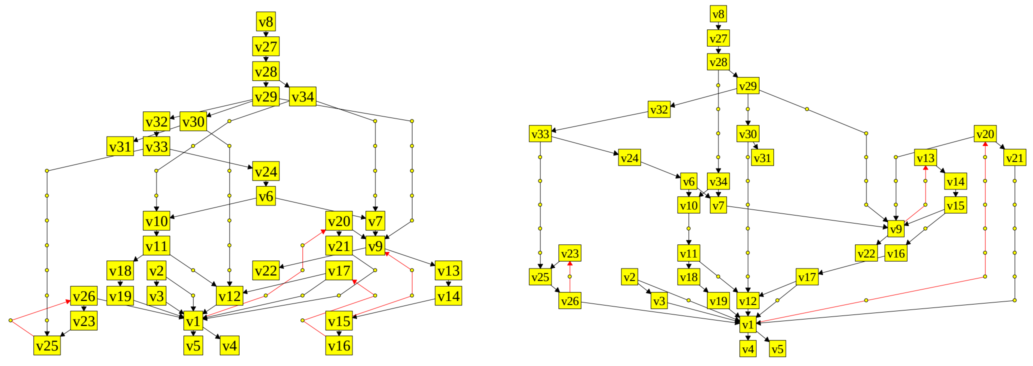
Table 6.12: Results of the experiment *B4*, for the graph *Worlddyn*.

Results for Gd94dir HQ		Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp	GA40 Comp
Initial Drawing		0	456	432	135	1008		
Subjects	1	0	455	407	129	952	99	87
	2	0	442	507	141	1134	100	100
	3	0	439	476	146	1083	100	100
	4	0	448	558	200	1260	100	99
	5	0	442	530	191	1176	100	100
	6	0	451	620	167	1495	99	99
	7	0	456	408	127	901	90	70
	8	0	452	424	133	972	99	98
	9	0	455	409	127	935	98	84
	10	0	455	429	134	954	94	81
	11	0	456	429	136	990	16	1
	12	0	452	435	142	1008	99	98
	13	0	456	432	132	1008	1	0
	14	0	443	430	134	1026	100	99
	15	0	455	424	136	1008	95	81
	16	0	439	469	134	1060	100	100
	17	0	453	409	135	952	99	96
	18	0	452	406	131	901	99	98
	19	0	455	429	137	936	94	81
	20	0	449	423	147	954	100	99
	Min	0	439	406	127	901	1	0
	Max	0	456	620	200	1495	100	100
	Av	0.00	450.25	452.70	142.95	1035.25	89.10	83.55
	StD	0.00	5.97	58.20	20.10	142.48	27.80	29.76
GA20	Best	0	451	410	143	952		
	50Best	0	456	414	131	952		
	Worst	0	456	432	133	1008		
	Min	0	451	404	126	952		
	Max	0	456	432	143	1008		
	Av	0.00	455.90	418.64	131.15	980.17		
	Std	0.00	0.54	9.50	2.41	28.02		
GA40	Best	0	443	410	135	952		
	50Best	0	456	408	129	952		
	Worst	0	456	432	129	1008		
	Min	0	443	404	125	952		
	Max	0	456	432	135	1008		
	Av	0.00	455.54	411.65	129.60	964.49		
	StD	0.00	1.50	7.56	1.98	23.29		

Table 6.13: Results of the experiment *B5*, for the graph *Gd94dir*.

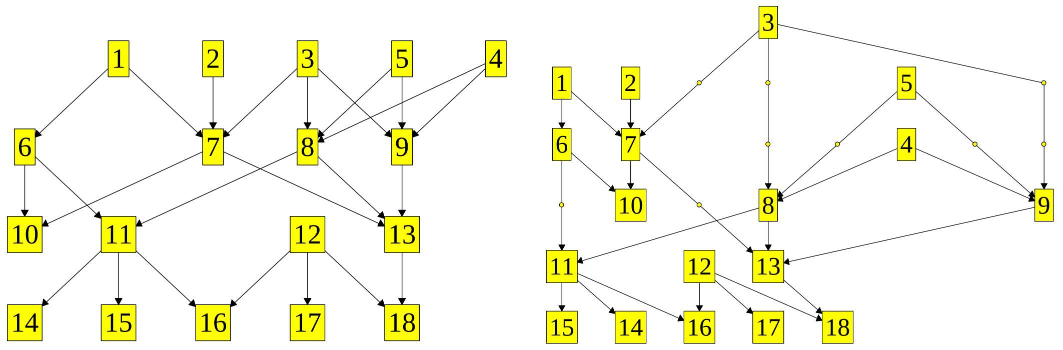
Subject	Level	Expected Knowledge	Sex	GA Time (s)	GA Exec	Interactive Time (s)	Total Performance
11	Honors	1	M	3878	52	9191	394
10	Master	1	M	5799	102	3423	466
2	Master	1	M	6221	49	4975	489
17	Honors	1	F	6843	97	6150	552
13	PhD	3	M	9493	129	3105	629
18	Honors	3	M	7418	121	5080	736
7	Master	1	F	10175	321	2152	790
14	PhD	2	M	5350	85	8167	798
1	Master	1	M	8552	101	3346	813
19	PhD	2	F	7312	262	4647	821
15	Honors	1	F	9729	193	3105	828
3	Master	1	M	8568	146	2609	835
16	Honors	3	M	7351	262	2895	867
8	Master	1	F	8861	204	2646	868
12	PhD	1	M	7056	232	4233	871
4	Master	1	F	9933	264	2435	945
5	PhD	2	M	6527	169	2152	969
20	Master	1	M	8249	104	3742	973
9	Honors	3	M	8250	450	3153	1018
6	Honors	3	M	7539	274	3800	1048
Correlation	**-**	**0.31**	**-**	**0.50**	**0.66**	**-0.54**	**1.00**

Table 6.14: Details about the subjects and their total performance.



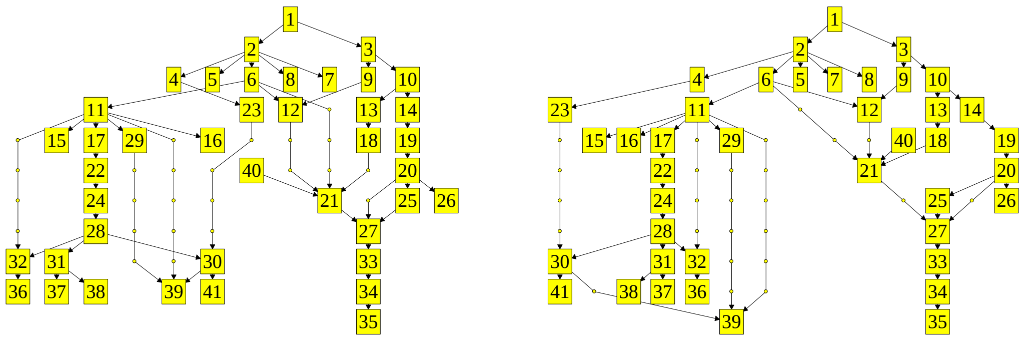
(a) (b)

Figure 6.8: Drawings of the *Csyntax* graph in experiment *A1*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 6.



(a) (b)

Figure 6.9: Drawings of the Klayer graph in experiment *A2*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 16.



(a) (b)

Figure 6.10: Drawings of the *Unixsys* graph in experiment *A3*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 9.

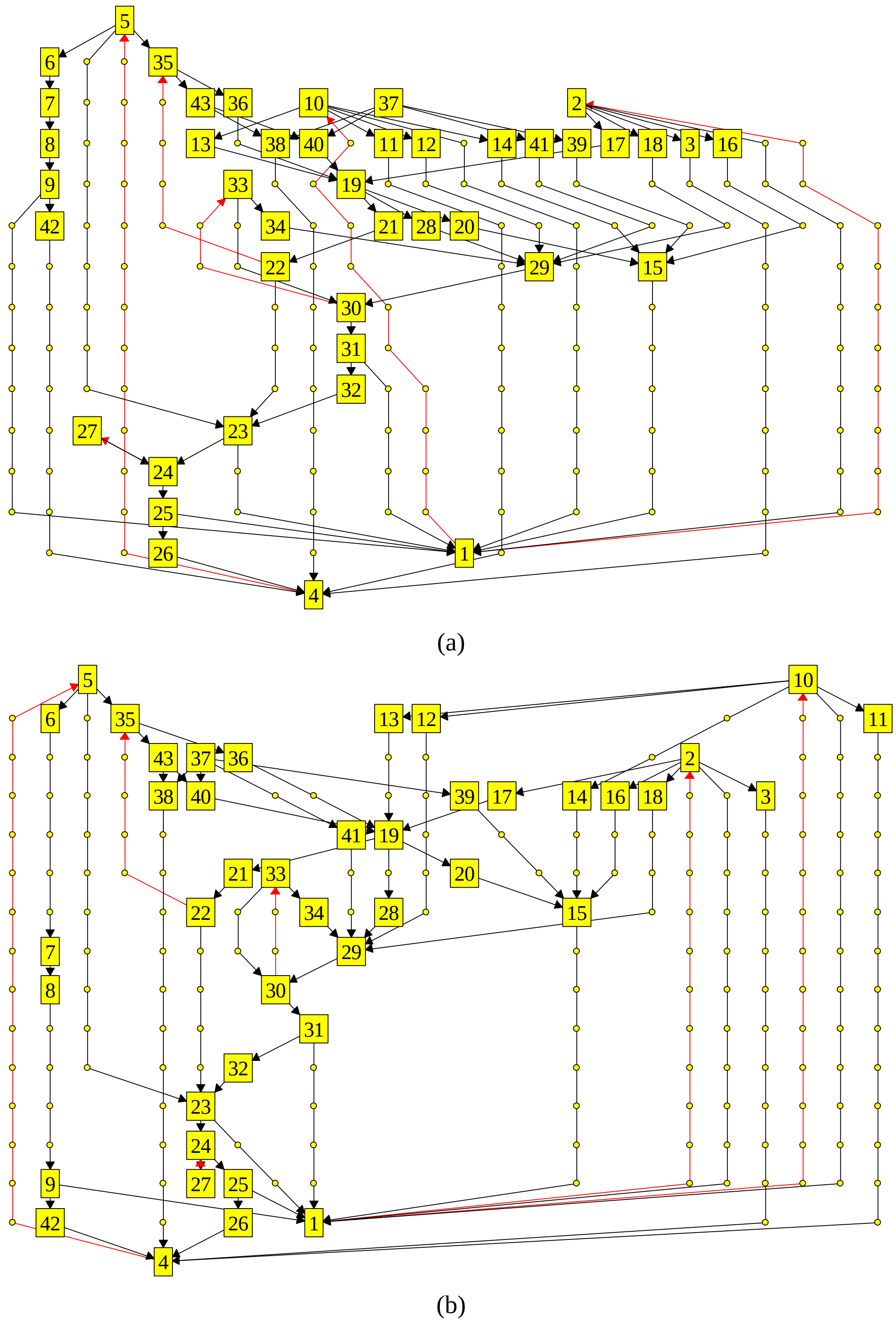
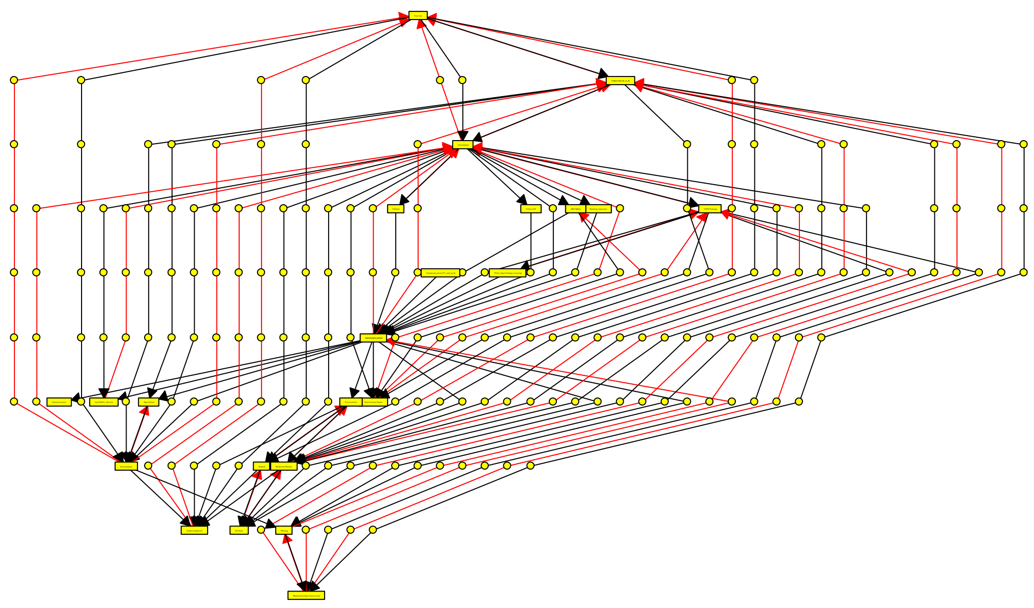
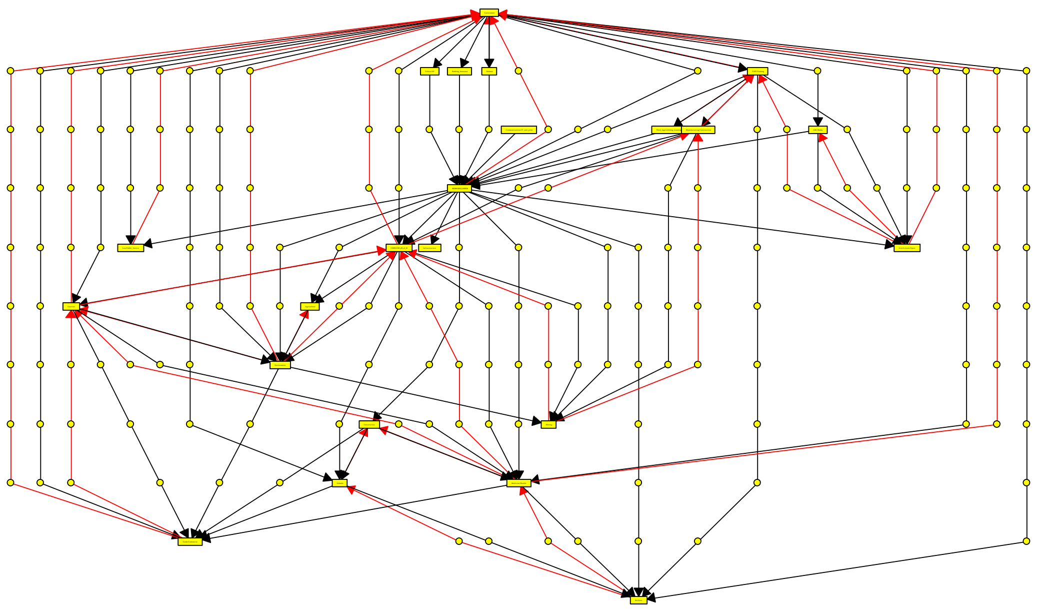


Figure 6.11: Drawings of the *Worlddyn* graph in experiment *A4*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 9.

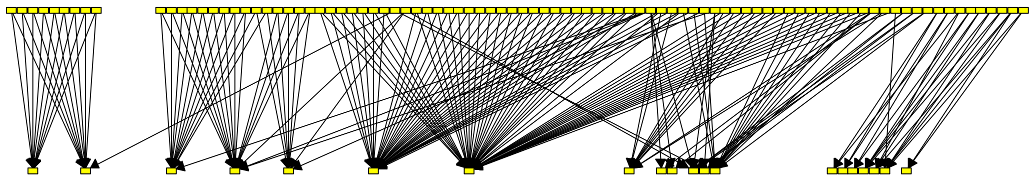


(a)



(b)

Figure 6.12: Drawings of the *Knation* graph in experiment *A5*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 9.


(a)

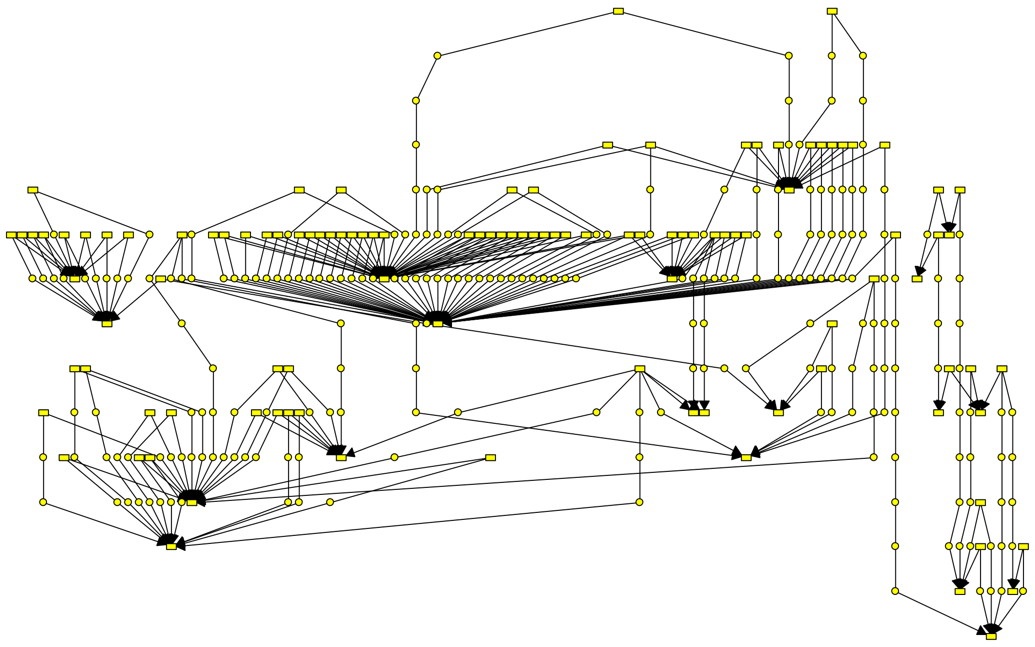
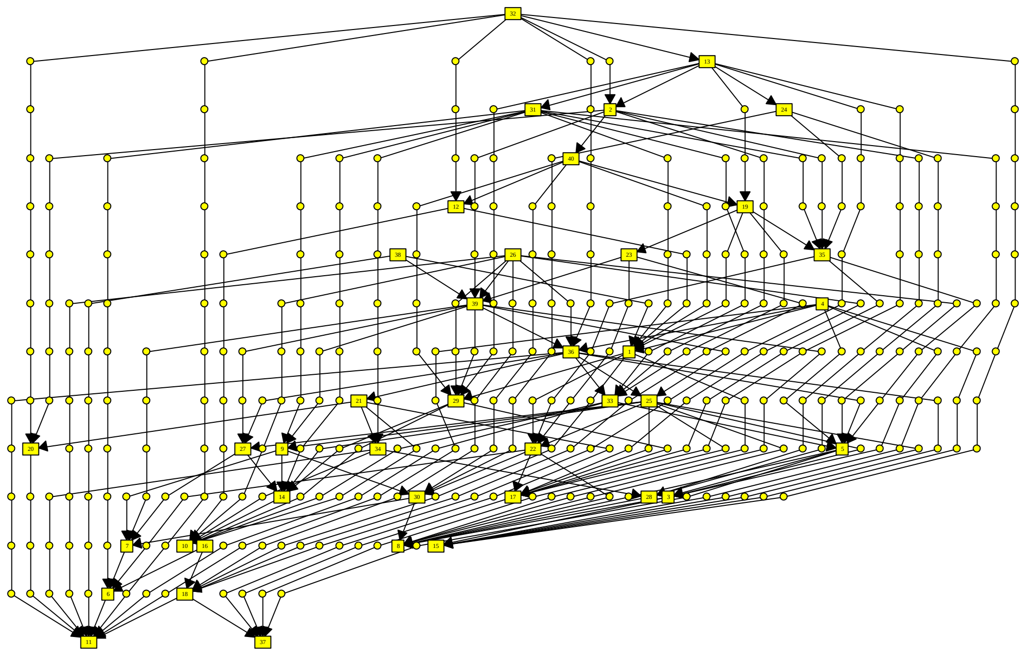

(b)

Figure 6.13: Drawings of the *Telcall* graph in experiment *A6*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 8.


(a)

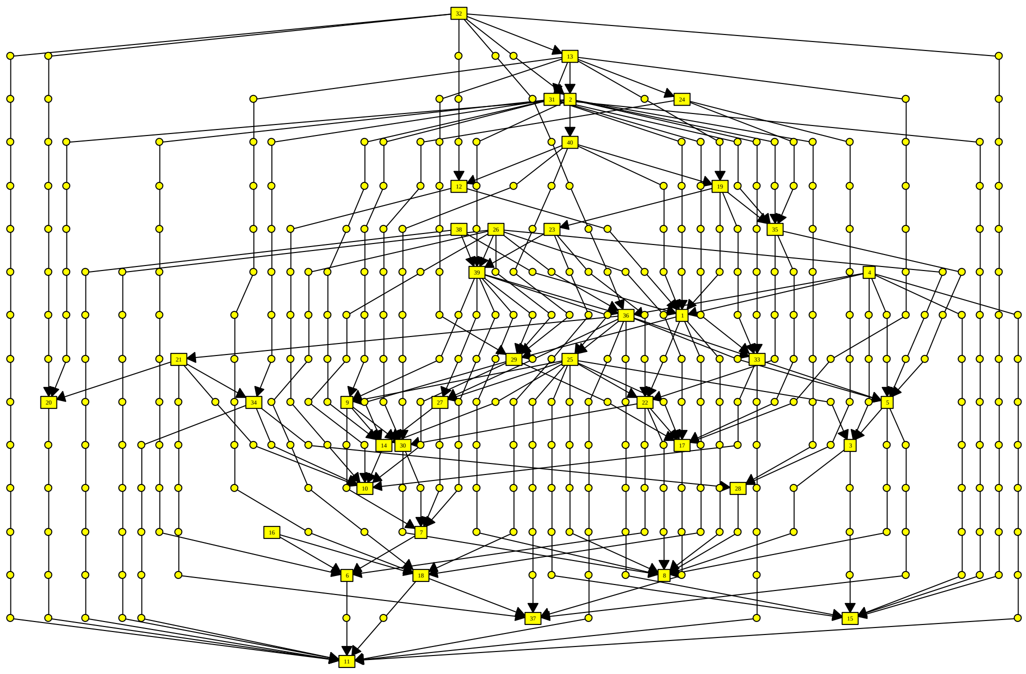

(b)

Figure 6.14: Drawings of the *Gd94dir* graph in experiment *A7*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 20.

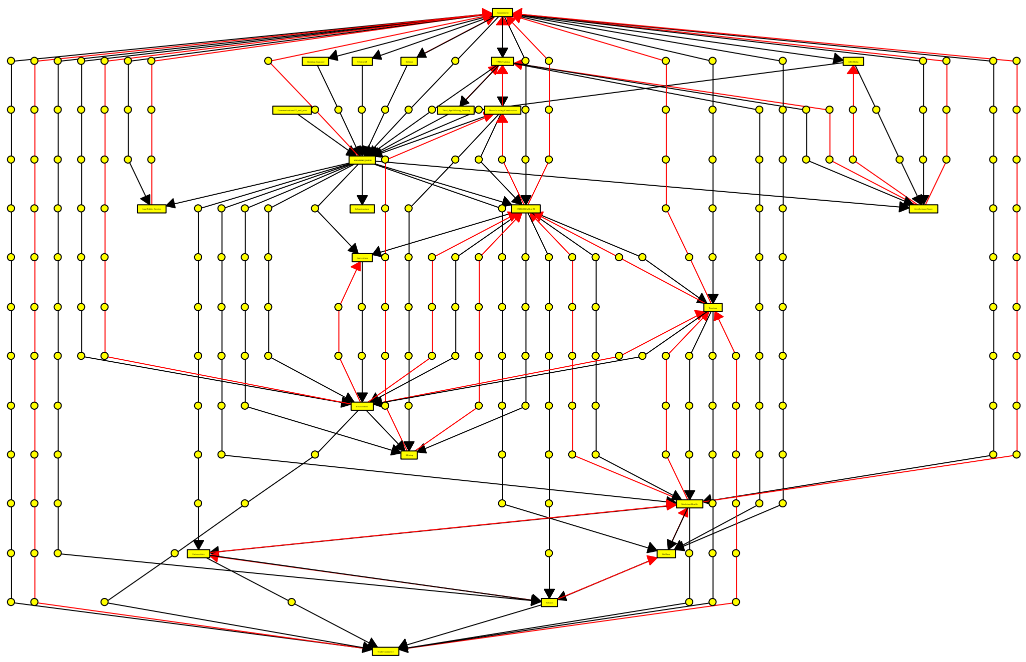


Figure 6.15: Drawing of the *Knation* graph created by Subject 20 in Experiment *B1*.

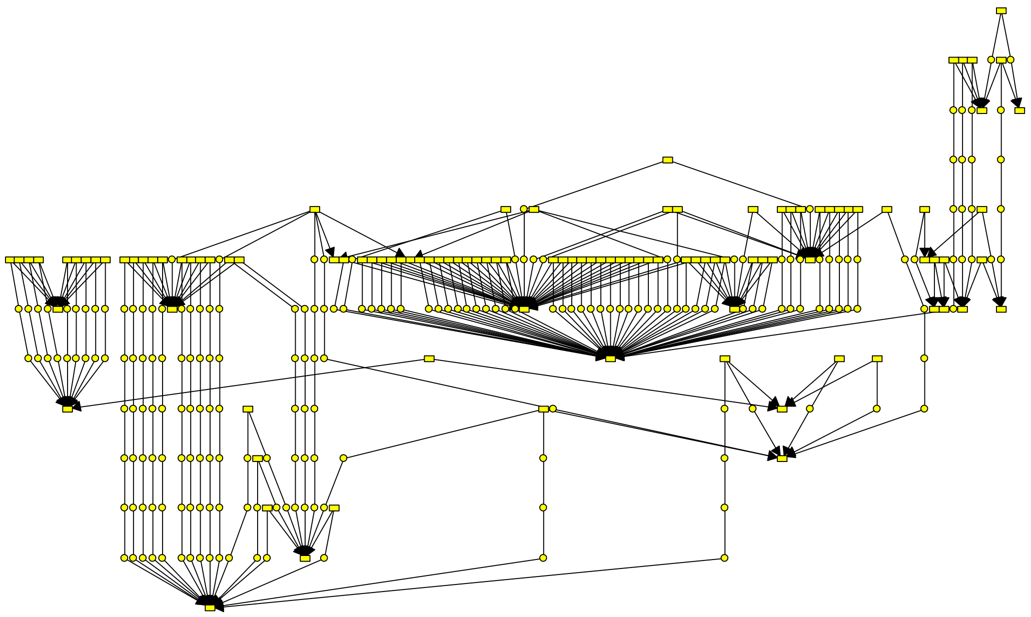
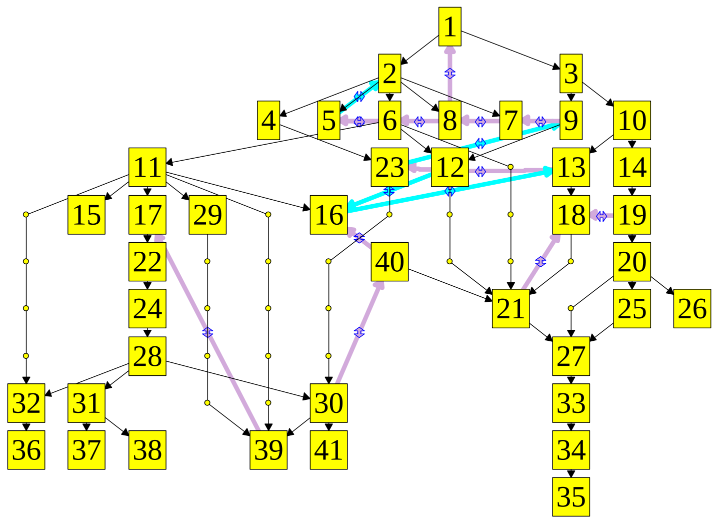
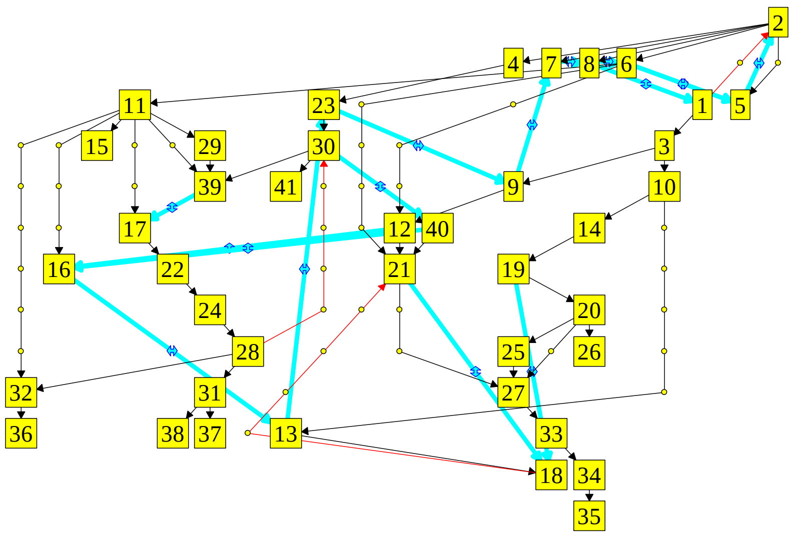


Figure 6.16: Drawing of the *Telcall* graph created by Subject 16 in experiment *B2*.

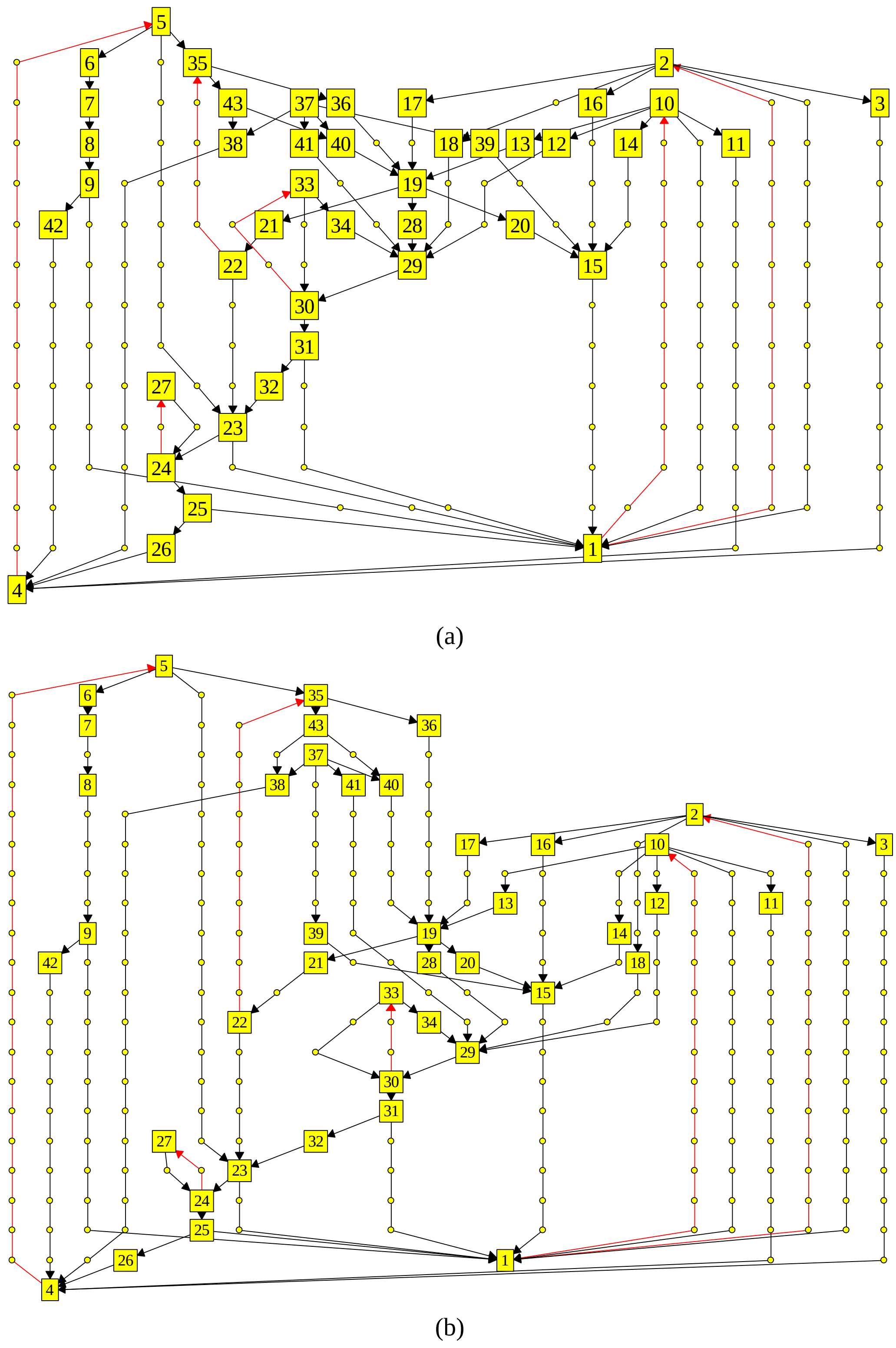


(a)



(b)

Figure 6.17: Drawings created in experiment *B3*, for the graph *Unixsys* with layout constraints. Figure (a) is the initial drawing. It has 12 unsatisfied constraints. Figure (b) is the improved drawing created by Subject 7, where all constraints are satisfied.


(a)

(b)

Figure 6.18: Drawings of the *Worlddy* graph in experiment *B4*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 5.

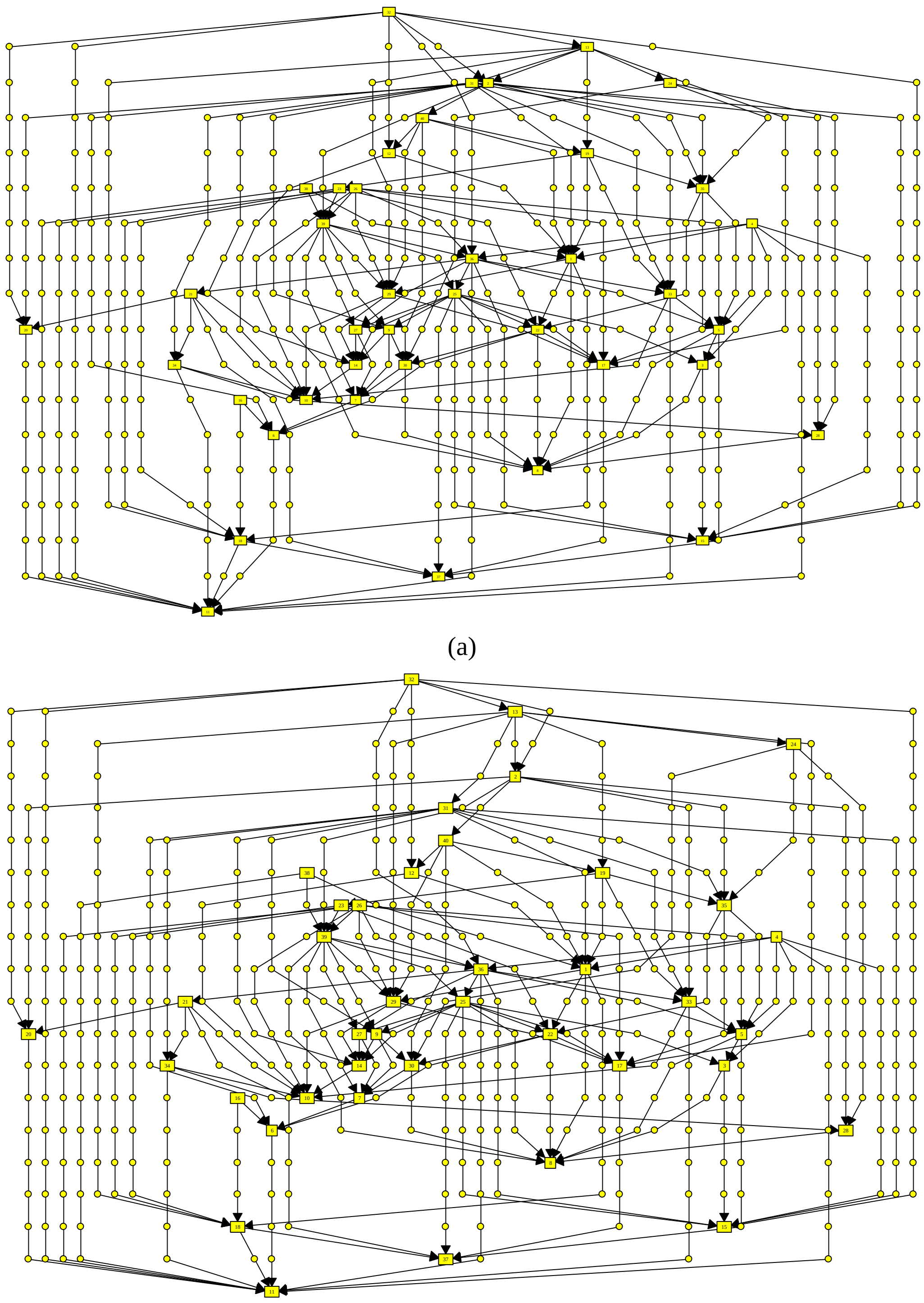


Figure 6.19: Drawings of the *Gd94dir* graph in experiment *B5*. Figure (a) is the initial drawing. Figure (b) is the improved drawing created by Subject 16.

6.6 Remarks

In this section we discuss several issues regarding the use of focus and layout constraints in the genetic algorithm and its internal structure. Suggestions for further implementations are also presented.

6.6.1 Focus and Constraints

Our genetic algorithm can be used in situations where an initial graph drawing simply needs to be improved, or when the structure of the graph changes dynamically and its drawing has to be updated.

For such needs, the focus facility provides an effective way of performing a drawing task. It allows the improvement of the desired regions of a graph drawing, while preserving the layout of areas which already have good quality (recall that areas not selected by the user are not changed).

Focus may also reduce computational resources necessary to produce an improved drawing. Processing time can be saved by exploring only the space of possible layouts for the selected vertices of the graph. The structure of the individuals in the genetic algorithm takes further advantage of this issue by keeping the minimal amount of information necessary to represent the selected elements. In addition, much processing time is saved by not evaluating the quality of the entire drawing when computing cost vectors for the individuals.

Note, however, that focus is beneficial only when the region to be redrawn is small compared to the entire drawing. Otherwise the merge procedure implemented in the genetic algorithm causes unnecessary overhead.

The support for layout constraints offers more control of the drawing activity. Constraints are useful, for example, for defining an ordering of the graph vertices without assigning precise coordinates to them. The approach that we used for supporting layout constraints in the genetic algorithm is reasonably flexible: it allows new types of constraints to be included by changing the objective function to contain a measure of the constraint satisfaction and, possibly, developing mutation and crossover operators to solve them. It would even be interesting to have new operators added to the current implementation of the genetic algorithm, as the existing operators do not guarantee that all constraints are satisfied.

6.6.2 Human Interaction

Human interaction with the genetic algorithm is necessary for two types of activities: (1) adjusting the system to produce a drawing with the characteristics that the user wants; and (2) helping the genetic algorithm to converge to a better drawing according to a predefined set of aesthetic criteria and constraints – whenever the method becomes stuck in bad local minima or takes to long to perform a particular improvement. In general, layout constraints, focus and manual changes play an important role in both tasks. However, the experiments with human interaction in the present and in the previous chapter investigate only the second type of activity. We did not run experiments where the users insert their subjective drawing aesthetic into the system, since most of subjects were not experts and it would also be hard to compare their results. Rather, we imposed a precise order for the aesthetic criteria, which is already common in many graph drawing applications. In the experiments we also fixed the set of layout constraints (in experiment *B3*), or allowed them to be changed, but investigated this facility as a tool for helping convergence (in Chapter 5).

Evidently, in real applications several interactive facilities should be available such as insertion of domain-dependent constraints and options for changing the set of aesthetic criteria. It is possible, for instance, to allow the users to specify a different priority order to the drawing aesthetics. Figure 6.20 shows an example where minimizing the number of edge crossings was set as more important than reducing the number of offending edges in the priority order. This modification was done manually in the code of the genetic algorithm, but it could be implemented as an option of the graphical interface. Playing with aesthetic criteria is an interesting way of learning how different measures affect the drawing of a graph; a user may even discover that a particular priority order provides a more meaningful drawing for a certain type of graph.

Code Optimizations

Since the genetic algorithm is used as an interactive tool, where the response time has to be short, optimization of the data structure and of the code of the algorithm becomes an essential issue. A reduction of processing time is already obtained by using focus, as discussed in the previous section. Nonetheless, we have implemented some other simple optimizations:

- We keep track of which vertices and edges are affected by the mutation or crossover operators. When computing the quality of an individual, we recalculate only the contribution of the affected elements to the cost vector.

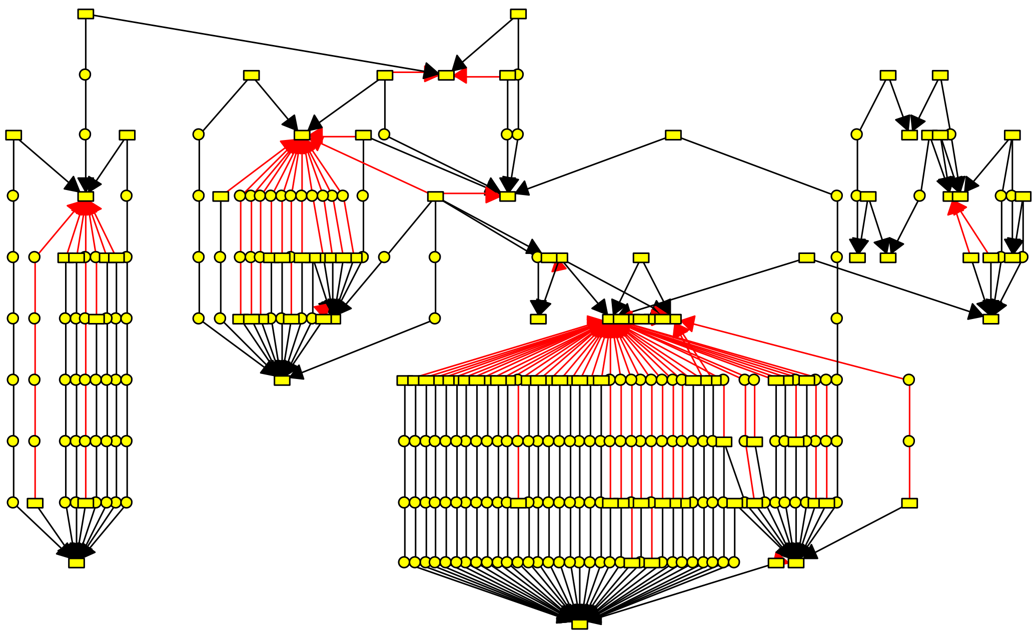


Figure 6.20: A drawing of the *Telcall* graph produced by minimizing the number of edge crossings as the most important aesthetic criterion.

- A bucket structure is created for the drawing D' and for every new individual. The structure classifies all vertices by their layers, so that a list of the vertices in a particular layer can be efficiently recovered.

Many other adjustments are possible. An interesting one, that we did not implement, is to recompute the parameters of the cost vectors only if and when necessary. For instance, suppose that we are doing a tournament selection with pairwise competition between two individuals, I_1 and I_2, with quality vector $Q_1 = (q_1^1, q_2^1, \ldots, q_7^1)$ and $Q_2 = (q_1^2, q_2^2, \ldots, q_7^2)$ respectively. If we find that $q_i^1 < q_i^2$ for an i, $1 \leq i \leq 6$, then we do not have to compute and compare the values of q_j^2 for all $j = i+1, i+2, \ldots, 7$, since the individual I_2 has already lost the competition and will be destroyed. This strategy is proposed by Rosete-Suárez *et al.* in [162]. The authors have presented results for graph drawing showing that the system with this optimization is much faster than an ordinary one.

Comments on the GDHints System

In the interview, run as part of our human experiments, the subjects provided insightful comments regarding their interaction with the GDHints system.

All subjects said that the interface was intuitive and easy to use. They did not mention about the interval of 3 seconds for updating the screen with the most recent drawing produced by the genetic algorithm. However, we do not think they felt this to be long. In fact, when a new best drawing is produced by the genetic algorithm, the system indicates this fact by playing a small animation. The animation itself takes about 3 seconds and, in some ways, "fills in" the time gap to the next update of the system.

Nevertheless, the subjects made several suggestions for new interactive facilities that could enhance the graph drawing task. The most promising ones were:

- Including an *undo* feature. This was mentioned by almost all subjects. Even though the system allows returning to the best drawing produced so far, such a drawing could differ from the current drawing by many interactive steps performed by the user. If the system returns to the best drawing instead of to the most immediate previous one, then much work may be lost.
- Improving the assignment of X-coordinates to new dummy vertices – which are created when a user manually moves the endpoints of an edge. At the moment, the system does not produce a necessarily good position to new dummy vertices. It may cause edge crossings that could be solved easily.
- Implementing a tool for automatically reversing the order of the X-coordinates of a group of vertices, when they are manually moved from the left to the right-hand side of the drawing or vice-versa. The goal here is to solve new edge crossings created by edges going from the moved vertices to their fixed neighbors in the drawing.
- Developing a tool for expanding the drawing (that is, shifting vertices in order to open empty columns and rows in a particular region of the drawing). This tool should preserve the quality of the drawing as much as possible.

6.6.3 The Internal Structure of the Genetic Algorithm

The decisions for choosing a different representation for the graph drawing solution and for implementing mutation operators in a particular way were based on several design and performance issues. We explain some of these decisions in this section.

Solution Representation

The genetic algorithm in [187] implements a different representation for the individuals, called *edge length representation*, that works only for directed acyclic graphs.

Instead of using the X, Y-coordinates of the vertices in the drawing, the *edge length representation* stores two values, $p(v)$ and $l(v)$, for every vertex v. The value $p(v)$ is a real number between 0 and 1 that determines the position of v within its layer. The value $l(v)$ is an integer number that determines the difference between the layer of v and the layer of the lowest predecessor of v. A drawing can be constructed from this representation by initially computing a topological ordering of the vertices of the graph. The source vertices are put in the highest layers. Then the remaining vertices are visited in topological order and are assigned to a layer according to their l-values and to the layer of their lowest predecessor. The X-coordinates of the vertices are computed by sorting all vertices in every layer by their p-values.

The *edge length representation* has the advantage of allowing adjustments of the position of several vertices by moving a single vertex. For instance, changing the p-value of a vertex v may result in a new ordering of the vertices in the same layer and, therefore, result in different X-coordinates for them. Changing the l-value of v may move automatically all successors of v some layers up or down. This effect helps the algorithm to escape from local minima, what would be difficult to do by independently moving just a few vertices at a time. Unfortunately, we cannot adopt the *edge length representation* in its totality in our genetic algorithm since we handle general directed graphs that may have cycles. In this case, there is no topological ordering.

As an alternative, we decided to use absolute X, Y coordinates in the individuals, and compensate the adjustment effect of the *edge length representation* by having vertex-overlap resolution and closure-based movements implemented in the basic routines.

Design of the Operators

We experimented with different setups for the operators of the genetic algorithm, and found that some options result in convergence to better solutions than others. For instance, the mutation *RANDOMCHANGE* without the linear distribution of probabilities (all offsets k have the same probability of being chosen) produces individuals with much lower quality. These individuals are difficult to improve and rarely propagate their characteristics to new generations.

The mutation *SOLVER* also gives better solutions than the system without it. This mutation provides a fast way of solving constraints and minimizing the number of offending edges, which are the elements with highest priorities in the quality measure of the individuals.

Moreover, treating vertex overlaps explicitly is important. Recall that we prevent overlap by shifting vertices in the routine *MOVEX*. An alternative approach would be to allow vertex overlap, but penalize it in the cost vector. However, this could result in individuals with many overlaps, that are destroyed before propagating their attributes to the next population; even though such individuals could be significantly improved by solving the overlaps as we do.

Finally, the choice of the values of the global Boolean variables *samesearchdir* and *sameshiftdir* affects the quality of the solutions as well. We observed that some values could help to escape from local minima. However, this depended strongly on the drawing under improvement and on the set of selected vertices. A general compromise choice was to set *samesearchdir* and *sameshiftdir* randomly before running any mutation or crossover operator.

6.6.4 Hill Climbing

We tried a different configuration for the evolutionary cycle of the genetic algorithm, where the crossover operator was deactivated and the population consisted of only one individual. The modified algorithm, that we call here HC, is essentially a Hill Climbing method, and can be described by the following code:

1. Let I be first individual, extracted from D'.
2. Repeat until a stop condition is satisfied:
   a) Choose a mutation operator randomly, and create a new individual I' by applying the operator to I.
   b) If I' is better than I then replace I with I'; otherwise, discard I'.

We tested the HC in a fully automatic mode with the graphs used in Section 6.5, as we did for the original genetic algorithm. The HC was executed 100 times for every graph, each execution taking 40 minutes. We employed the same computers used in the human experiments.

Tables 6.15 and 6.16 show the summarized results of the HC for 20 and 40 minutes, respectively. The qualitative parameters of the drawings produced by the best, the fiftieth best (the median) and the worst executions are presented. Column *GA20Comp* in Table 6.15, and column *GA40Comp* in Table 6.16 are the same parameters used in Tables 6.2 to 6.13. Both columns give the number of

Results of the HC		Constr.	Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA20 Comp
Experiment *A1* - *Csyntax* graph								
HC20	Best	0	3	4	67	4	315	99
	50Best	0	3	6	54	4	252	53
	Worst	0	3	9	34	7	182	1
Experiment *A2* - *Klayer* graph								
HC20	Best	0	0	2	4	1	35	100
	50Best	0	0	3	8	1	54	87
	Worst	0	0	3	12	3	70	86
Experiment *A3* - *Unixsys* graph								
HC20	Best	0	0	1	27	4	143	99
	50Best	0	0	3	24	3	143	66
	Worst	0	0	3	24	6	132	0
Experiment *A4* - *Worlddyn* graph								
HC20	Best	0	6	21	152	26	390	100
	50Best	0	6	33	146	29	435	62
	Worst	0	6	42	150	25	375	2
Experiment *A5* - *Knation* graph								
HC20	Best	0	28	85	197	79	492	99
	50Best	0	28	124	347	102	950	63
	Worst	0	29	148	269	94	630	0
Experiment *A6* - *Telcall* graph								
HC20	Best	0	0	292	412	141	2360	99
	50Best	0	0	383	221	99	1166	67
	Worst	0	0	598	305	150	1545	4
Experiment *A7* - *Gd94dir* graph								
HC20	Best	0	0	452	395	125	901	100
	50Best	0	0	512	450	134	972	58
	Worst	0	0	543	335	114	795	0
Experiment *B3* - *Unixsys* graph with constraints								
HC20	Best	0	3	1	55	9	375	100
	50Best	1	3	9	65	12	323	51
	Worst	2	6	7	33	6	204	1
Experiment *B4* - *Worlddyn* graph with a high quality initial drawing								
HC20	Best	0	6	20	174	25	468	6
	50Best	0	6	22	166	25	425	0
	Worst	0	6	22	166	25	425	0
Experiment *B5* - *Gd94dir* graph with a high quality initial drawing								
HC20	Best	0	0	443	419	132	1008	100
	50Best	0	0	456	408	132	952	69
	Worst	0	0	456	432	133	1008	0

Table 6.15: Results produced by the Hill Climbing for 20 minutes.

executions (between 1 to 100) of the original genetic algorithm that resulted in a worse drawing than the one considered in the corresponding row of the table, for the same amount of processing time.

The results obtained with the HC are very promising. It performed as well as the genetic algorithm when considering the quality of the drawing in the best execution. Sometimes the HC was

Results of the HC		Constr.	Offend. Edges	Cross.	Dummy Vertices	Bends	Area	GA40 Comp
Experiment *A1* - *Csyntax* graph								
HC40	Best	0	3	4	67	4	315	99
	50Best	0	3	6	52	5	198	54
	Worst	0	3	9	34	7	182	1
Experiment *A2* - *Klayer* graph								
HC40	Best	0	0	2	4	1	35	100
	50Best	0	0	3	8	1	54	87
	Worst	0	0	3	12	3	70	86
Experiment *A3* - *Unixsys* graph								
HC40	Best	0	0	1	27	4	143	99
	50Best	0	0	3	24	3	143	66
	Worst	0	0	3	24	6	132	0
Experiment *A4* - *Worlddyn* graph								
HC40	Best	0	6	18	161	29	416	100
	50Best	0	6	32	150	20	400	67
	Worst	0	6	39	155	24	416	4
Experiment *A5* - *Knation* graph								
HC40	Best	0	28	84	253	86	645	99
	50Best	0	28	121	481	94	1173	64
	Worst	0	28	182	289	82	765	0
Experiment *A6* - *Telcall* graph								
HC40	Best	0	0	276	353	121	2360	100
	50Best	0	0	376	217	91	1166	61
	Worst	0	0	597	278	134	1590	1
Experiment *A7* - *Gd94dir* graph								
HC40	Best	0	0	452	395	121	901	99
	50Best	0	0	505	365	123	848	71
	Worst	0	0	535	472	147	1007	2
Experiment *B3* - *Unixsys* graph with constraints								
HC40	Best	0	3	1	55	9	375	100
	50Best	1	3	9	53	10	300	52
	Worst	2	6	7	33	6	204	1
Experiment *B4* - *Worlddyn* graph with a high quality initial drawing								
HC40	Best	0	6	20	174	25	468	1
	50Best	0	6	22	166	25	425	0
	Worst	0	6	22	166	25	425	0
Experiment *B5* - *Gd94dir* graph with a high quality initial drawing								
HC40	Best	0	0	440	404	128	952	100
	50Best	0	0	456	408	128	952	51
	Worst	0	0	456	426	130	1008	5

Table 6.16: Results produced by the Hill Climbing for 40 minutes.

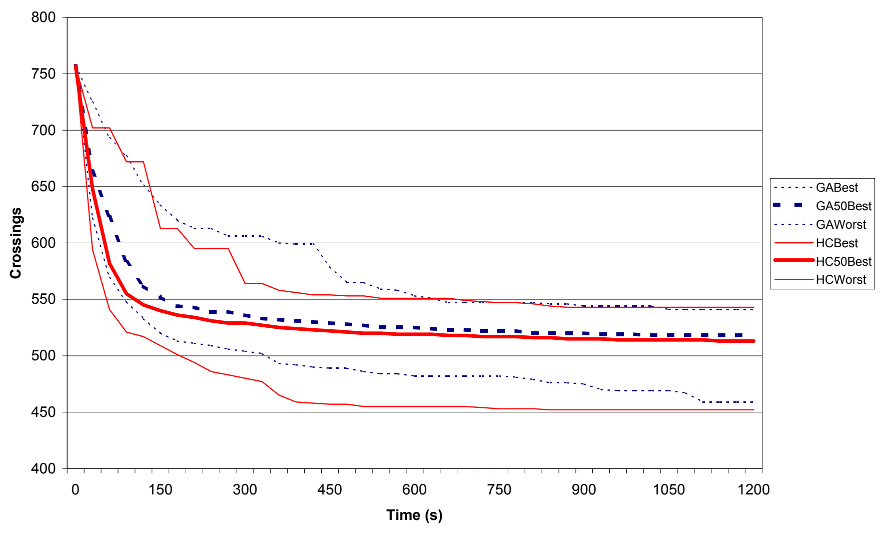


Figure 6.21: Improvements of the *Gd94dir* graph by the Hill Climbing and the genetic algorithm.

slightly worse, sometimes it was better[6]. The median HC, however, was better than the median genetic algorithm in almost all cases. This is indicated by values of *GA20Comp* and *GA40Comp* grater or equal to 50 in the rows *50Best*. The only case where the HC performed poorly was for the *Worlddyn* graph with a high quality initial drawing. Our explanation for this is that improving such drawing demands moving out of a local minimum, what the HC obviously cannot do. On the other hand, the HC was significantly better than the genetic algorithm for the *Klayer* graph. This suggests that the genetic algorithm may have fallen into a local minimum that was harder to escape, or simply could not explore the local space of solutions sufficiently in the given amount of time.

The main advantage of the HC over the genetic algorithm is that it can improve the drawings faster. Figure 6.21 presents the reduction of the number of edge crossings for the *Gd94dir* graph over time, performed by the HC and the genetic algorithm. The chart shows the quality of the best, median and worst executions (among 100 executions for each algorithm) taken every 30 seconds. The number of offending edges was zero during the entire computation.

[6]Some of the improvements of the HC over the genetic algorithm were for less significant aesthetic criteria such as the number of dummy nodes and bends. In those cases, the number of offending edges and edge crossings could not be reduced.

Our results are in agreement with the findings of Rosete-Suárez *et al.* [161], which also show that a simple stochastic Hill Climbing method for graph drawing outperforms a genetic algorithm.

Unfortunately, we did not use the HC in our human experiments since its test was completed only after the evaluation of the genetic algorithm. Nevertheless, we believe that the positive results with the HC have important implications for interactive optimization tasks. The first aspect is that by using a faster algorithm the optimization process can be sped up and completed in less time. In addition, human interaction and visual feedback can also benefit: the user does not have to wait long for a solution refresh (which was set for 3 seconds using our genetic algorithm), since more significant improvements of the drawings occurs per period of time.

Note that the user is still necessary, as the results of the HC are worse than the human-generated solutions for most of the experiments. The HC also does not replace the genetic algorithm totally since the Hill Climbing approach cannot escape from local minima. The genetic algorithm may be particularly useful when no further improvement can be obtained with the HC.

The HC is promising, and it needs further investigation.

CHAPTER 7

User Hints for Map Labeling

In Chapters 5 and 6 we applied the User Hints framework to improve the search for better solutions in terms of a fixed set of quality measures. In the present chapter, we concentrate mainly on how to refine an optimization problem (as discussed in Chapter 2), so that domain knowledge can be included. We show this process for the problem of labeling point features in cartographic maps. An interactive framework is introduced that allows users to change the objective function and the constraints of the problem intuitively. The framework can also be used to help convergence to optimal solutions.

Part of this work was published at the proceedings of the Twenty-Sixth Australasian Computer Science Conference, in Adelaide-Australia, in February, 2003 [51].

The remainder of the chapter is organized as follows: Section 7.1 introduces the Map Labeling problem and techniques for solving it. Section 7.2 explains the need for having human-computer interaction in Map Labeling processes. Section 7.3 presents a sequence of steps that characterizes an automatic Map Labeling approach. In Section 7.4 we introduce an interactive framework based on User Hints that extends the automatic labeling steps. Section 7.5 describes a system that we built for testing our interactive Map Labeling framework. Section 7.6 presents an evaluation of the system and the framework with experts in Cartography. Finally, Section 7.7 discusses other issues related to experts' heuristics, data structure and memory management, and extensions to the Map Labeling framework.

7.1 The Map Labeling Problem and Automatic Methods

The *Map Labeling Problem* consists of assigning positions for the labels of the graphical features of a map, so that these elements can be identified. Three types of graphical features are normally considered: *point features*, *area features* and *line features*. Point features are usually represented by a dot, a circle or a small icon. Cities in a country map are examples of point features. Area features are larger graphical objects, mostly polygonal areas. Typical area features are political regions (such as states) in a map. Line features are lines that may vary in thickness, and are used to describe rivers, roads, streets and train lines. Figure 7.1 illustrates the labeling of features for a map of Australia. The labels are assigned to points representing cities and to areas indicating states and other geographic regions[1].

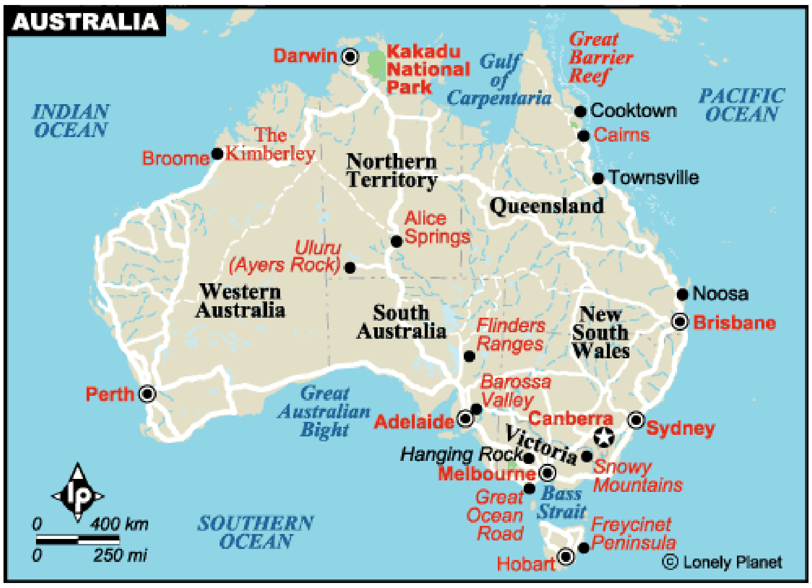


Figure 7.1: Map of Australia with labeled features from the Lonely Planet. (c) Copyright 2003 Lonely Planet Publications. All rights reserved. Used with Permission. *www.lonelyplanet.com*.

The Map Labeling Problem is well-known in Cartography and had its scientific foundation established in the second half of the last century, most notably with the work of Imhof [96, 97] and Yoeli [201]. Imhof and Yoeli defined a set of rules that are used up to the present day to guide the development of labeling algorithms; these are:

1. **readability**: labels must have legible sizes;
2. **unambiguity**: each label must be easily identified with exactly one graphical feature; and
3. **avoidance of overlaps**: labels should not overlap with other labels or other graphical features.

[1]The association between types of features and objects of the map is in some ways flexible. A graphical element usually treated as a particular type of feature may be represented by another type. For instance, suburbs in maps of cities can be represented by point features, when the picture is scaled down so that the area becomes small. Note as well that each type of feature in general implies a different standard for label placement. While point features have their labels usually displayed around them, area features have the labels placed inside them. Line features are usually labeled by placing the label text along a line or inside it, if the line is thick.

Since the debut of Map Labeling as a scientific area, this problem has increased in importance and has been the subject of a considerable number of publications. An illustrative chart of the distribution of the Map Labeling publications over the last forty five years can be seen in the *Map-Labeling Bibliography Web Site* [198]. The Computational Geometry Impact Task Force [32] has even classified the Map Labeling problem as an important issue to be investigated by future research.

Current map-labeling research has several applications that involve not only the development of Cartographic and Geographic Information Systems in general, but also related products such as tools for Information Visualization [56, 108] and for image annotation [172]. Whenever there is a need for assigning graphical labels to graphical objects, Map Labeling techniques can be useful.

Techniques for solving Map Labeling problems commonly adopt a model for label placement. Three main models have been proposed in the literature [143], and are illustrated in Figure 7.2 for point features (labels are represented abstractly by rectangular boxes):

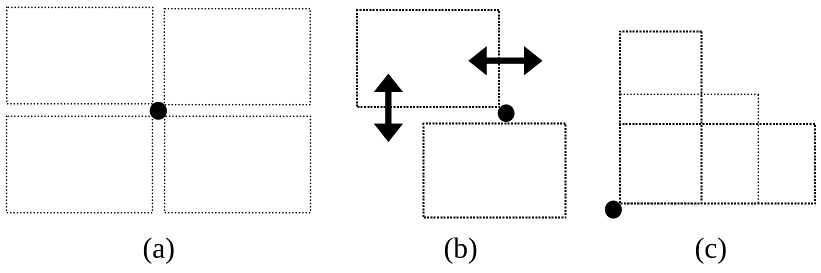


Figure 7.2: Label position models.

- **Fixed-Position Model**. The labels can only be placed on a finite set of candidate positions defined around each feature. This is the most popular model. It has special cases such as the 2-Position and 4-Position Fixed models, which specify two and four positions respectively for every feature.

- **Slider Model**. In the Slider model (Figure 7.2(b)) the labels can slide continuously in one or more directions while still touching their features. This movement allows a fine grain placement where labels can be shifted left, right, up or down in order to avoid overlaps.

- **Fixed-Position Scalable Model**. It is similar to the Fixed-Position model, where the labels can only be placed in a predefined set of candidate positions. However, the size of the labels can be scaled in order to solve overlaps. A particular case of the Scalable model is the **Elastic Model** (Figure 7.2(c)), where the labels can be scaled, but their area should stay constant. The labels also have to stay attached to their point features.

These models provide a formalization for the labeling problem so that it can be solved computationally. For instance, by using the Fixed-Position model we can define a combinatorial optimization problem: assigning labels to candidate positions, so that all features are labeled and there is no label-label or label-feature overlap. Figure 7.3 illustrates the 4-Position-Fixed model for labeling four point features representing cities. Note that some positions overlap with each other. The labeling task aims to choose one candidate position for every feature so that, when placing the label on that position, no overlap occurs. For cases where there is no assignment that yields an overlap-free labeling, one or more features must stay unlabeled. It is also possible to define preferences for the candidate positions in order to have the labels placed as much as possible on a particular side of their features (e.g: on the top-right corner of the feature). Preferences can be represented by costs assigned to the candidate positions. When including costs, the optimization problem needs to be redefined to computing an overlap-free assignment of labels to positions with minimum total cost (the sum of the costs of all label positions).

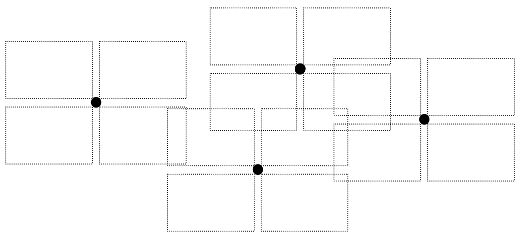


Figure 7.3: Candidate label positions for four cities. Each city is represented by a dot and has four candidate positions (represented by dotted rectangles around the feature) for label placement.

Based on the labeling models described above, four basic Map Labeling problems can be formulated[2]:

- **Decision Problem** – verifying whether there is a label assignment, such that all features are labeled using one of their candidate positions and no two labels overlap.
- **Label Problem** – if the answer to the Decision Problem is 'yes', then finding the labeling assignment without overlaps.
- **Number Maximization Problem** – finding a labeling assignment where as many features as possible are labeled without overlaps.

[2]These problems are usually related to the Fixed-Position model, but they can consider the Slider and the Scalable models.

- **Size Maximization Problem** – finding a maximum scaling factor s and a labeling assignment, such that all features are labeled, the size of the labels are scaled by s, and there is no label overlap.

These optimization problems are computationally difficult to solve for most practical cases [130]. For example, the decision problem for the 4-Position-Fixed model where labels are squares of equal size is *NP*-complete [70, 130][3]. The Label Problem, the Number Maximization Problem and the Size Maximization Problem depend on the Decision Problem. Therefore, they are NP-hard.

The minimization of the costs of the labeling solution is usually done in combination with the basic problems.

Several optimization methods have been proposed for Map Labeling, such as Expert Systems, greedy heuristics, Discrete Gradient Descent methods, Simulated Annealing and Integer Linear Programming.

Expert systems were presented initially as a promising approach [4, 54, 55, 72, 101, 147]. It was possible to describe the overlapping relation between labels as a set of logic rules, and use expert systems to obtain a labeling assignment. Nevertheless, it was later realized that the combinatorial nature of the labeling problem and the naive search implemented by many expert systems made it impossible to solve labeling instances of practical size [204].

Greedy heuristics and Discrete Gradient Descent methods for map labeling [34] were also investigated for some time. Greedy heuristics take the map features in a particular order, and construct a labeling solution by deciding about the labeling positions sequentially without backtracking. Discrete Gradient Descent methods are in general more “intelligent” approaches. An example of a Discrete Gradient Descent algorithm consists of starting with an unlabeled solution, and computing the costs of using every labeling position for every feature; it then chooses the feature and the position that yields a solution with minimum cost. This process repeats until no further improvement of the labeling can be obtained.

Greedy heuristics and Discrete Gradient Descent are simple to implement and fast to execute. However, they easily fall into local minima with poor global quality [34].

The problems presented by the previous algorithms are overcome by the Simulated Annealing proposed by Christensen *et al.* [33, 34]. This method provides very good results in a reasonable time. Moreover, a compromise between solution quality and processing time can be achieved by

[3]For the 2-Position-Fixed model, the problem can be solved in polynomial time on the number of candidate positions. This can be proved by reducing it to a 2SAT problem [70, 95].

adjusting the annealing schedule of the algorithm.

Simulated Annealing is, in fact, one of the most successful methods for map labeling. It has been extensively employed also as a benchmark for evaluating new map labeling techniques.

Another successful method is based on Integer Linear Programming. Zoraster [202, 203] formulated an Integer Linear Programming model for map labeling, and used Lagrangian Relaxation to solve it. Strijk, Verweij and Aardal [176, 192] presented later an Integer Linear Programming model with several optimizations that can quickly solve large instances, with up to 950 point features in the 4-Position-Fixed model, using Branch-and-Bound.

The work of Strijk *et al.*, Zoraster [203] and Christensen *et al.* [202, 203], as well as many other approaches, are strongly related to modeling the map labeling problem as a Maximum Independent Set problem, and then solving it using some standard techniques. This process will be explained in more detail later.

Other map labeling methods also of importance are Approximation Algorithms (mainly for some restricted problems) [11, 19, 148, 154], Tabu search [200], and Genetic Algorithms [45, 155, 188, 189, 191].

7.2 The Need for User Intervention

Even though an extensive literature on Map Labeling is available [198], it is interesting to note that these scientific results do not entirely satisfy the needs of real map labeling tasks performed in industry. A technological gap exists here between scientific research and real-world applications. While existing optimization methods are suitable for some particular map labeling aspects, such as the avoidance of overlaps and ambiguity[4], and preferences for label placement, they very often neglect other important issues.

For instance, labeling methods usually do not deal with domain knowledge about the semantics of the labels. See, for example, the case in Figure 7.4 where three features are labeled as "St. George", "Washington" and "Happy Bay" respectively. Considered in this order the labeling may cause the user to misread the labels to say "George Washington" or "George Washington Happy". This effect can occur even though there is sufficient space in between the labels. The human map labeler may prefer to minimize this problem by positioning the label "Washington" above its feature. Such an operation can be easily performed by hand. However, it would be difficult to encode this

[4]Ambiguity is resolved by most map labeling approaches by ensuring a minimal distance between pairs of labels and between labels and features that are not directly associated.

knowledge as a general constraint in an optimization tool, since it depends on the semantics of the labels.

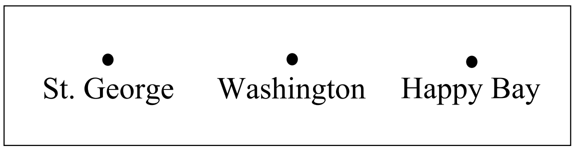


Figure 7.4: Semantic problems with labels.

Another difficulty is that map labeling approaches solve labeling problems globally, providing a labeling solution that tends to be good in general, but that may sacrifice the quality of some particular areas of interest for the cartographers.

Furthermore, it seems hard to cope with the large amount of constraints and aesthetics that the cartographers naturally use when producing maps by hand. This includes knowledge about the implicit importance of some features, rules for breaking down long labels in small lines, alternative positions for labeling features in dense areas, and consistency with rules used in other maps. Such information varies from cartographer to cartographer, and is based on experience acquired from working in the field.

For all these reasons, Map Labeling in industry is still dependent on a large amount of human work. Optimization methods for map labeling do not deal with the type of domain knowledge described above. The user has to adjust the solution in a post-processing stage by manually moving the labels. This technological gap can be seen, for example, in the main companies in Australia that produce maps and street directories – Ausway and UBD. Both companies use systems that automatically produce initial labeling solutions. The computer-generated solutions are helpful, but they are in general not totally pleasing to the cartographers. Thus, a manual activity is necessary to improve them [80, 164, 185].

Unfortunately, manual adjustment of a labeling solution can also be a complex and very time-consuming process, particularly for dense regions of the map where the replacement of one label can cause new problems with the surrounding text.

These difficulties suggest that Map Labeling can benefit from interactive optimization. Therefore, we investigate in the present chapter a framework based on User Hints where cartographers can interact with automatic map labeling techniques. The goal is still to allow the user to adjust the labeling solutions in order to convey domain knowledge. However, we aim to reduce the amount of manual work by providing a better integration with automatic labeling methods. We concentrate our study on the Maximum Number Labeling problem with the Fixed-Position model. Only point features are considered and labels are represented by rectangles of unit height.

In the next section, we divide the labeling process into a sequence of small steps. These steps are used later as a basis for our interactive Map Labeling framework.

7.3 Labeling Steps

We define the map labeling process as a sequence of steps that goes from inputting graphical information about features and labels of the map to modeling the labeling task as an optimization problem, and finally to producing a labeling solution. Figure 7.5 shows a diagram of the labeling steps used in the present chapter. This division in steps is very intuitive and is done by many map labeling approaches such as the ones presented by Edmondson *et al.* [61], Zoraster [203], and Strijk *et al.* [176].

The labeling process starts with a list of point features containing their coordinates and their labels. A set of graphic attributes is then assigned to the labels, describing the type and the size of the font used for writing the text. In the next labeling step, a labeling model is incorporated into the process. We use the Fixed-Position model described before, where a number of predefined candidate label-positions are assigned to each point feature. The candidate positions are rectangular regions in the map. The size of the regions depends on the label text and on the font size. Each candidate position has a cost value (a real number in the interval $[0, 1]$), that represents the user preference for placing a label in it; the higher the cost value, the lower the preference.

Some basic notations are useful for formalizing these first steps: let $F = \{f_1, f_2, \dots, f_n\}$ be a set of n point-features of a map labeling problem. Every feature f_i has a textual label l_i of dimensions $r_i = (w_i, h_i)$, and is assigned a set π_i of candidate positions for displaying l_i on the map, for $i = 1, 2, \dots, n$. We call $P = \bigcup_{i=1}^{n} \pi_i$ the set of all candidate positions. The function $\lambda(p) \in [0, 1]$ gives a *cost value* for each $p \in P$.

The next labeling step is to define a combinatorial model [61, 176, 203]: we construct an undirected *conflict graph* $G = (V, E)$ describing all overlaps between pairs of candidate label positions, and use it for computing a label assignment. The conflict graph is as follows: for each candidate position p in P, there is a vertex v_p in V; for every pair of label positions p and q in P that overlap on the map, we add an edge (v_p, v_q) to E; if a candidate label position p assigned to a point feature $f_i \in F$ overlaps another point feature $f_j \in F$, with $i \neq j$, then we insert an edge (v_p, v_q) in E for

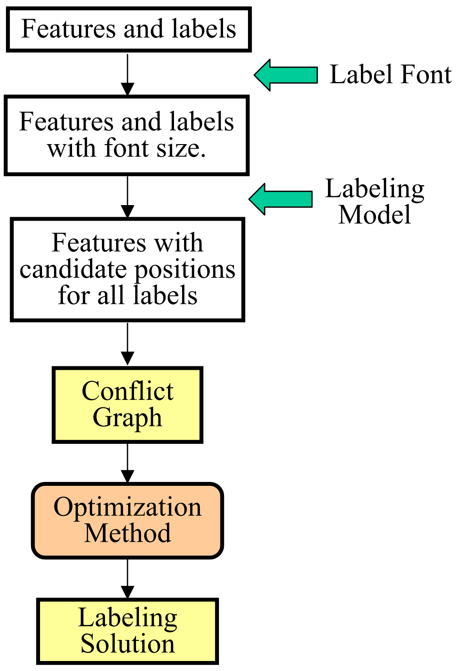


Figure 7.5: The Map Labeling steps.

every candidate label position q in π_j. Moreover, for every feature $f \in F$, the vertices associated to its candidate label positions form a clique in G. The cost values of the candidate positions can be extended to the conflict graph by defining $\lambda'(v_p) = \lambda(p)$ for all p in P.

An example of a conflict graph is shown in Figure 7.6 for the candidate positions presented in Figure 7.3. Many of the user interactions presented in this chapter manipulate the conflict graph, as we explain in Section 7.4.1.

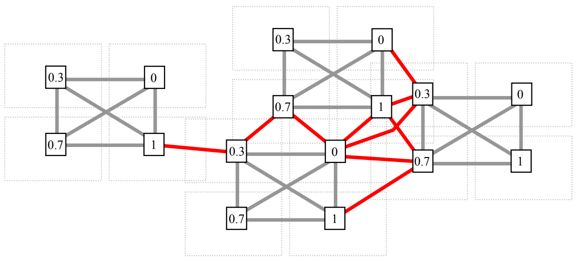


Figure 7.6: The conflict graph of the candidate label positions in Figure 7.3. Vertices, representing label positions, are drawn as squares. The original candidate label positions, shown as dotted rectangles, are presented here only for reference. Each candidate position has a cost assigned to it. Light-color edges connect vertices associated to the same feature. Dark-color edges indicate overlaps on the map between two candidate positions.

After the conflict graph has been constructed, the next labeling step is to solve an optimization problem. The most common approach (see for example Strijk *et al.* [176]) is to solve a variant of the Minimum Cost Maximum Independent Set Problem (MCMISP) for the conflict graph. A set $I \subseteq V$ is *independent* if no two vertices of I are joined by an edge in E. The set $I \subseteq V$ is a *maximum independent set* if it is independent and it has maximum size. The *Minimum Cost Maximum Independent Set Problem* is to find a maximum independent set I for which the sum of the costs of the vertices in I is minimized.

A solution for the MCMISP is directly associated with a solution for the labeling problem, since the vertices in I indicate candidate positions for labeling the features. Note that the edges in the conflict graph are chosen so that if I is independent, then the candidate positions associated with the vertices in I do not overlap. If no vertex related to a particular feature $f \in F$ appears in I, then f remains unlabeled. Figure 7.7 shows the maximum independent set with minimum cost for the conflict graph in Figure 7.6, and the corresponding labeling solution.

In the rest of this chapter we refer to a set of vertices $I \subseteq V$ and its associated candidate positions indistinguishably as a *labeling solution*.

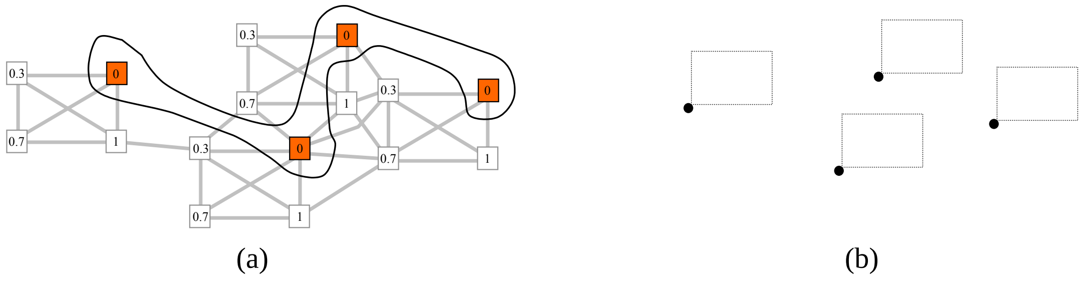


Figure 7.7: (a) the maximum independent set with minimum cost for the conflict graph in Figure 7.3, and (b) its corresponding labeling.

The MCMISP provides a clear representation for the map labeling problem. However, in some circumstances (for example, for interactive optimization methods) we need a more flexible model. Christensen *et al.* [33] present a model where label overlaps are tolerated, but they are penalized in the objective function. The problem consists of minimizing the number of labels taking part in overlaps plus the number of unlabeled features. A labeling solution in this case may have overlaps, but it is expected to be an intermediate stage to a better labeling.

We use Christensen's model and modify the conflict graph to include an *unlabeled vertex* ϵ_i for every feature f_i, $i = 1, 2, \ldots, n$. The unlabeled vertex has a very large cost and forms a clique with the other vertices associated to the same feature. If an unlabeled vertex appears in I, then we do not label its feature.

The new optimization problem is formalized as finding a subset $I \subseteq V$, such that $|\{p \in P :$

$v_p \in I, (v_p, u) \in E, u \in I\}| + |\{\epsilon_i \in I : 1 \leq i \leq n\}|$ is minimized, and the solution I is constrained to have exactly one vertex associated to every feature. The sum of the costs of the unlabeled vertices in I is a secondary criterion for minimization.

Christensen *et al.* propose a data structure for computing and maintaining the cost of a labeling solution efficiently. For every candidate position, we record a list of all other candidate positions (associated to different features) and features that intersect with it. Let p_i be the candidate position in use by a feature f_i, with $p_i \in (\pi_i \cup \epsilon)$, and $p_i = \epsilon$ indicating that f_i is unlabeled, $1 \leq i \leq n$. For every feature f_i, $1 \leq i \leq n$, we store a counter of the number of pairwise overlaps between p_i and all other features, and between p_i and p_j, for $1 \leq j \leq n$ and $i \neq j$. If $p_i = \epsilon$ then the counter for f_i is zero.

The total number of labels taking part in overlaps is given by the number of features whose counter is different from zero. If a feature f_i, $1 \leq i \leq n$, has its label candidate position in use p_i changed, then the counter for f_i and for other related features can be updated by looking at the lists of intersecting elements associated to the previous and the new candidate positions referred by p_i. Whenever a counter slips from (to) zero, the total number of overlapping labels is incremented (decremented) by one.

Note that the labeling steps described here characterize a fully automatic labeling approach. Next we consider a framework where the human plays a role.

7.4 An Interactive Map Labeling Framework

Our interactive framework for Map Labeling provides the following resources:

1. Mechanisms for refining a labeling problem – the user can adjust the constraints and objectives of the problem so that the optimum solution for the modified problem has the properties in which the user is interested. Manual changes and other interaction facilities offer an intuitive way for inserting domain knowledge into the optimization process.

2. Automatic optimization methods for incremental improvement of a labeling solution – instead of performing time-demanding manual adjustments of labeling solutions, the user can take advantage of automatic optimization methods. The methods compute a solution considering recent domain knowledge entered into the system, and support focus.

3. Quality feedback – visualizations provide feedback to the user about the quality of the current

labeling solution and the progress of the optimization process.

We implement these resources by extending the automatic labeling steps presented in the previous section. The extension is shown in Figure 7.8. The first modification is to allow a continuous processing between the optimization method and the labeling solution; this is indicated by the cycle between the last two boxes on the left-hand side of the figure. In addition, data structures used in each labeling step are extended to become persistent and dynamic. Parts of the data structures can also be selected, so that interactive and automatic operations are executed only on the selected elements. Selection is shown by a vertical line that intersects the labeling steps. Finally, the user can control the labeling steps through a number of interaction facilities.

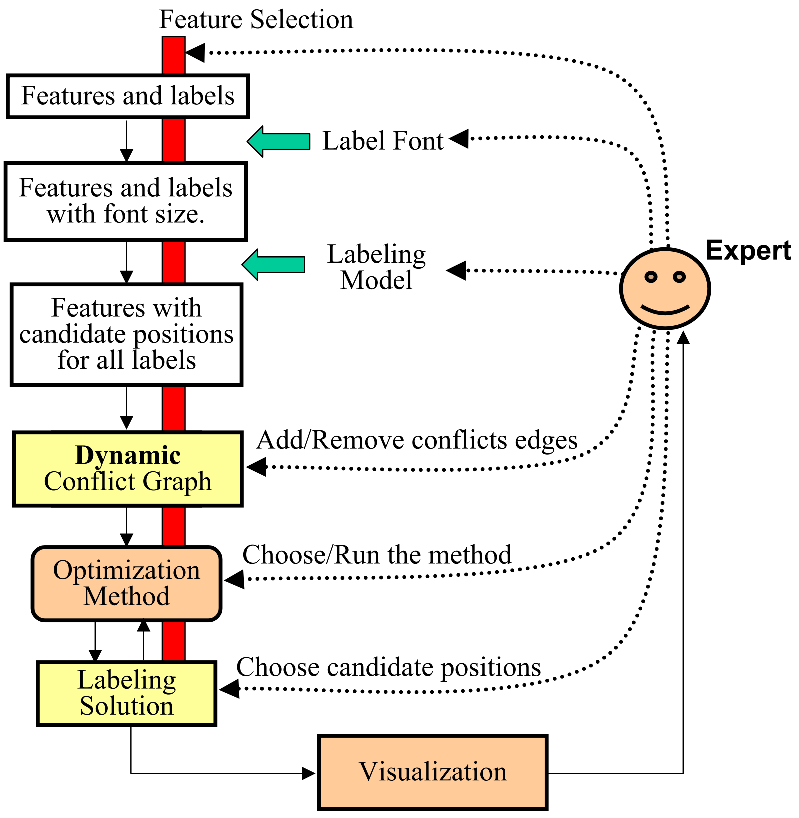


Figure 7.8: The interactive framework for Map Labeling.

The framework employs data structures for keeping a set F of point features, the label l_i and the set π_i of candidate positions for every feature $f_i \in F$, the conflict graph $G = (V, E)$ and a labeling solution $I \subseteq V$ (that functions as a working solution). These data structures are persistent – they are kept in memory rather than discarded after running a labeling method, in contrast to what is done by traditional map labeling approaches – so that changes made by the user are preserved.

Selection is defined for point features, for candidate positions and for vertices of the conflict graph. However, user direct control of the selection mechanism is allowed only for point features. When the user selects a point feature, its corresponding candidate positions and vertices in the conflict graph are automatically selected. Deselecting a point feature causes the selection of its candidate positions and associated vertices to be undone.

A visualization tool provides pictures of the labeling. The user can tune the visualization as we explain in more detail later. The framework also includes the best solution agent (implicit in Figure 7.8), which saves the best labeling solution produced so far.

7.4.1 Interactions

Several interaction facilities are investigated in our framework. We present a complete list of them below (in this subsection and in the remainder of this chapter we use the formal notations introduced in Section 7.3):

a) Changing the font size of the label of a feature.

b) Activating/deactivating a predefined candidate position p for a feature – this action causes the insertion/removal of a vertex v_p into/from the conflict graph, plus its related edges.

c) Creating a customized candidate position for a particular feature. The new candidate position can be located anywhere in the map – this action also causes the inclusion of a new vertex and of some edges in the conflict graph.

d) Changing the cost of a candidate label position – the change is made on the associated vertex in the conflict graph as well.

e) Directly creating a new edge in the conflict graph or removing an existing one.

f) Constraining a feature f to be labeled – this action temporarily deletes the unlabeled vertex assigned to f from the conflict graph.

g) Constraining a feature f not only to be labeled, but also to use a particular candidate label position p – in this case, all vertices in the conflict graph related to f are temporarily removed, except the vertex v_p; the related edges are also deleted.

h) Choosing and running a different optimization method, when more than one algorithm is available.

i) Focusing the optimization method on a group of selected features – when an optimization method is executed, only the selected features can have their labeling assignment modified.

j) Manually changing the labeling solution computed by the optimization method – this operation is implemented by allowing the user to swap a vertex p in I with another vertex q, where p and q are associated with the same feature.

All these operations are done graphically and in an intuitive way through a graphical interface.

Changes performed by the user in a particular step of the labeling process are immediately propagated to the data structures of the later steps. For instance, increasing the font size of the label of a feature causes all related candidate positions to enlarge. New overlaps may then occur, forcing new edges to be added to the conflict graph. Similarly, if the user deactivates a candidate position p of a feature $f \in F$ whose vertex v_p is in use by the labeling solution I, then v_p is removed from the conflict graph and from I. The unlabeled vertex of f is added to I instead.

Note that the selection mechanism not only allows the user to focus the optimization method on a particular section of the labeling problem, but also to restrain all other interaction facilities to this region. For example, by calling the appropriate tool for activating/deactivating candidate positions causes a modification only of the selected features. This is quite different from the traditional map labeling approach, where a labeling model is defined uniformly for all features.

Even though the diagram for user interaction in Figure 7.8 seems different from the User Hints framework in Chapter 3, it matches the types of user hints we described perfectly. The interaction facilities (a) to (g) represent adjustments to the objective function and to the constraints of the problem, since they affect the conflict graph used as an input to the optimization methods. Facility (h) corresponds to controlling the labeling method directly, while facility (i) is a focus mechanism. Facility (j) is a typical manual adjustment of the solution.

With these interaction facilities the user has sufficient flexibility to refine the problem and improve a map labeling solution. He or she can either execute a fully manual postprocessing of a computer-generated solution, or combine human intervention with automatic optimization methods. Examples of how to use the framework for solving specific labeling problems are given in Section 7.4.5.

7.4.2 Solution Quality

We say that a feature is *conflicting* if its associated vertex v in I is not an unlabeled vertex, and if there is an edge $(v, u) \in E$ with $u \in I$ and u is also not an unlabeled vertex. In other words, the feature is conflicting if it is labeled and its current label position is overlapping another label or feature, or if the feature itself is overlapped by the label position in use by another feature.

The *quality* of a labeling solution I, for a conflict graph $G = (V, E)$ and set of features F, with $I \subseteq V$, is defined by $Q_G(I) = (q_1(I), q_2(I))$, where:

- q_1 is the number of conflicting features implied by I plus the number of unlabeled vertices in I.
- q_2 is the sum of the cost of all non-unlabeled vertices in I.

Given two labeling solutions I_1 and I_2 for the same labeling problem, I_1 is *better* the I_2 if $Q_G(I_1) < Q_G(I_2)$ in lexicographic order. The values for q_1 and q_2 can be efficiently computed using the data structure described in [61].

7.4.3 Optimization Methods and Focus

Any optimization method that is able to solve the map labeling problem described in Section 7.3 can be used in our framework. This includes practically all methods mentioned in this chapter such as Simulated Annealing, simple greedy heuristics, Discrete Gradient Descent, Integer Linear Programming[5] and Genetic Algorithms. The optimization methods must input the set of features, the conflict graph and, optionally, an existing labeling solution to the problem. As output, they must produce a new labeling solution.

The existing labeling solution can be a starting point for producing a better labeling via an improvement process. It also provides a bound for the quality of any new labeling.

When integrating the optimization methods in the framework, we have to consider feature selection (focus) and constraints. Constraints can be easily implemented (as described previously, in Section 7.4.1) by translating them into changes of the conflict graph before running the methods. Focus may be implemented here in two possible ways:

1) Hard-coding focus in the optimization methods. In this case, the algorithms are modified to remove from or add to the existing solution I only selected vertices of the conflict graph. If

[5]provided that there is a feasible solution.

the existing labeling solution contains non-selected vertices, then these are preserved in the new solution.

2) Executing a preprocessing stage where a simplified labeling problem is created containing only the necessary information for labeling the focused area. The optimization methods are not changed. They work on the simplified problem, and the resultant solution is integrated with the existing solution for the whole labeling.

Support for focus is relatively easy to hard-code in general improvement methods, such as greedy heuristics and Simulated Annealing. A list of selected features is sufficient to control which elements of the existing solution can be modified. Integer Linear Programming methods also permit a simple hard-code implementation: integer constraints can be defined for assigning the labels of the non-selected features to their current candidate positions in I. This means that only the variables associated to the selected features are free to change.

The preprocessing option, however, offers several advantages: it allows the use of existing map labeling algorithms without modification; it simplifies the labeling problem by discarding several non-selected features; and it is also simple to implement.

In the next section, we explain the preprocessing approach in detail.

The Preprocessing

Given a list of features F, a conflict graph $G = (V, E)$, an existing labeling solution $I \subseteq V$, and a function $\sigma(V) \to Boolean$ that indicates whether a vertex $v \in V$ is selected or not ($\sigma(v) = true$ if v is selected, and $false$ otherwise), we compute a simplified labeling problem with features $F' \subseteq F$ by defining a *simplified conflict graph* $G' = (V', E')$, and a *simplified labeling solution* $I' \subseteq I$ as follows:

1. $V' = \{u \in V : \sigma(u) = true\ or\ ((u, v) \in E, u \in I\ and\ \sigma(v) = true)\}$,
2. $E' = \{(u, v) \in E : u, v \in V'\ and\ (\sigma(u) = true\ or\ \sigma(v) = true)\}$;
3. $I' = I \bigcap V'$.

The new set F' consists of all features of F that have at least one of their associated vertices in V'. Figure 7.9 shows the result of a preprocessing for a labeling problem involving three features in the 4-Position-Fixed model. The middle feature is selected and, consequently, all its vertices in the conflict graph. Note that the simplified labeling problem contains only the elements that may affect the quality of labeling assignments for the selected feature (Feature 2). The new problem does not include all candidate positions for Feature 3, since this feature is not selected and, therefore, its label position does not change. Furthermore, no labeling assignment for Feature 2 causes overlaps with the label of Feature 1; consequently, Feature 1 is not included in the simplified problem.

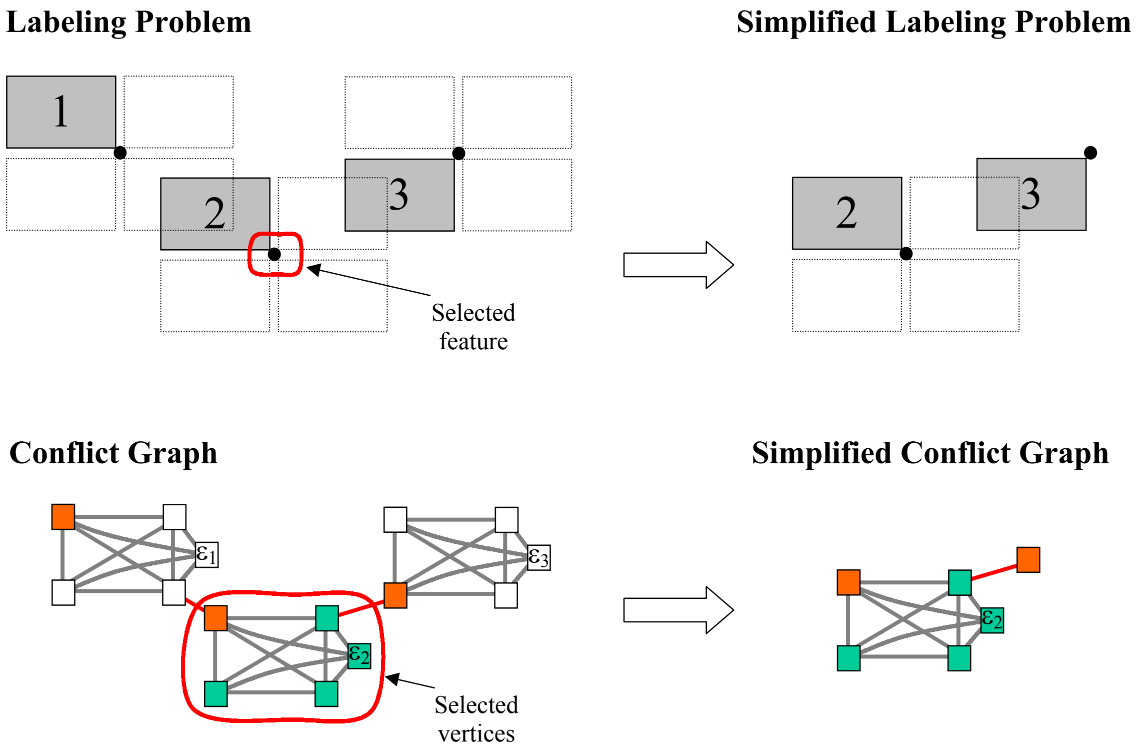


Figure 7.9: Preprocessing of a selected labeling problem.

A solution for the simplified problem provides a solution for selected features of the original problem. In addition, the labeling assignment for the non-selected features in F' (that is, Feature 3 in Figure 7.9) is preserved, since V' does not contain unlabeled vertices for these features, and all features are constrained to have exactly one of their associated vertices in I'.

7.4.4 Visualizations

Visualizations are provided to show qualitative aspects of the labeling solution and/or combinatorial characteristics of the problem.

The user can swap between two visualizations:

1) A geographical map with features and labels. This view has small variations that can be

chosen by the user, such as displaying a box around the labels and presenting all candidate positions. Figure 7.10 illustrates this visualization.

2) The conflict graph with all vertices in I highlighted. This is shown in Figure 7.11

Overlaps and unlabeled features are highlighted in the pictures by using icons with different colors and shapes.

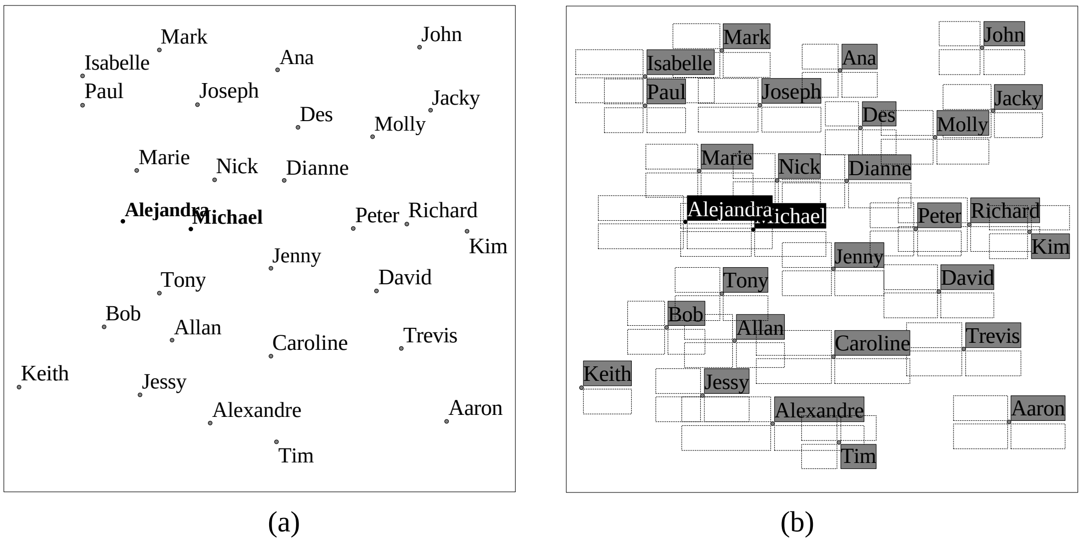


Figure 7.10: The geographic map visualization for a set of features with proper names. The labels "Alejandra" and "Michael" are overlapping. Two variations are presented: a traditional view (a), and a picture with all candidate positions that emphasizes the position in use (b).

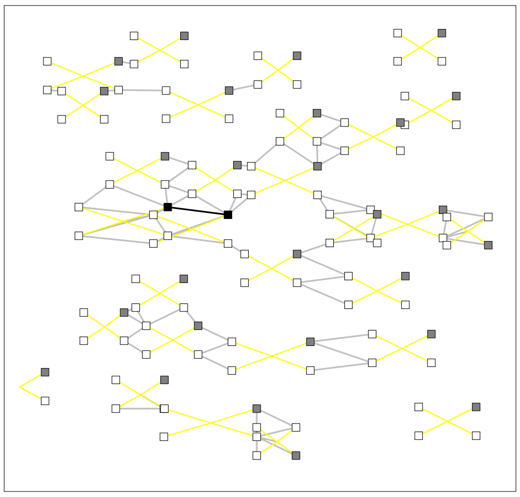


Figure 7.11: The conflict graph visualization for the labeling problem in Figure 7.10. The yellow lines (in light color) represent in fact a clique between vertices associated to a same feature.

In order to avoid overwhelming the visualization with too many details, we adopt an *information-*

demand policy: the user can control the amount of extra information to be displayed. This is done by combining the visualization tool with the feature-selection mechanism, so that extra details are shown only for the selected features. For instance, the user can hide all unselected unlabeled features or their candidate positions of all unselected features.

7.4.5 Labeling Improvement

In this section we explain how the interaction facilities can be used for solving overlaps and ambiguity, and for converging to better local minima.

Overlaps

In general, the optimization methods provide an initial labeling solution without overlaps, but with some unlabeled features. Overlaps are then created by the user when moving a label to a different candidate position, increasing the font size of a label, or changing the label text. Such changes may be based on domain knowledge, but they create problems that need to be solved.

Overlaps can be treated by selecting a group of the features including the conflicting ones, and re-executing an optimization method for reorganizing the labels. If there is no complete labeling without overlaps, then one or more features will be set to an unlabeled state.

Ambiguity and Semantic Problems

Ambiguity involves uncertainty in uniquely identifying to which feature a label is related. Figure 7.12 shows a crowded labeling region with ambiguous situations. For instance, it is difficult to identify the features for "Kotto" and "Osyth". The observer may also be unsure, at a first glance, about the features associated with labels "Slotnick" and "Tallbot".

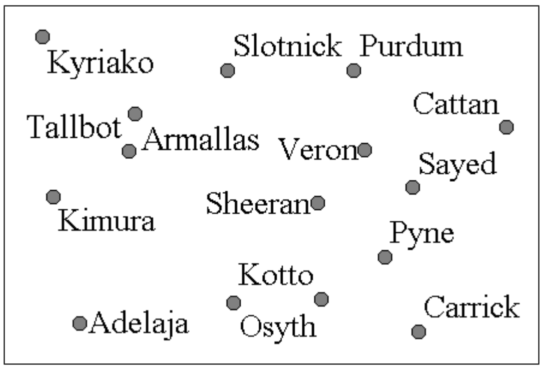


Figure 7.12: Ambiguous labeling cases.

Our interactive framework offers the user three semi-automatic ways of solving ambiguity:

1) Deactivating one of the candidate label positions in use that is causing ambiguity.

2) Increasing the cost assigned to one or both of the ambiguous candidate positions.

3) Or creating a "virtual" overlap by inserting an edge into the conflict graph that connects the pair of vertices representing the ambiguous candidate positions.

These operations can be performed using a graphical interface. After the operation, the user calls an optimization algorithm for computing a new labeling solution for the affected area.

The virtual overlap seems the most powerful approach. It prohibits the ambiguous case, and still allows some particular solutions that are not possible using the other two alternatives. For example, the virtual overlap allows the optimization method to resolve ambiguity by setting any of the features as unlabeled, and leaving the other feature unchanged; the other approaches do not have the same flexibility.

The three approaches can also be applied to semantic problems, as the one illustrated in Figure 7.4. For that problem, the user may create a conflict edge between labels "St. George" and "Washington", and between "Washington" and "Happy Bay", in order to force them to be repositioned.

Unlabeled Features

If the optimization methods tend to leave important features unlabeled, then the user can force them to be always labeled by defining labeling constraints visually. Figure 7.13 shows examples of such constraints, which are represented by pins. A pin can be assigned to a feature (saying that it has to be labeled) or to a particular candidate position (that has necessarily to be used).

When calling a method again, all labeling constraints are automatically translated into changes of the conflict graph (as explained in Section 7.4.1), and force the constrained features to be labeled.

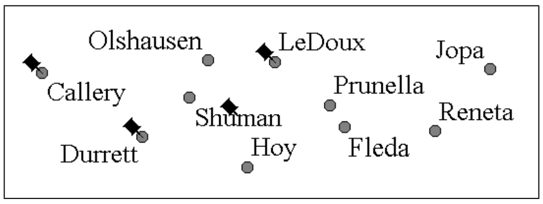


Figure 7.13: Pinned (constrained) features.

Other Resources

In practical cartographic processes, the user commonly improves the labeling of features by moving labels by hand. This is used to solve any labeling problem. We also support the same type of interaction, but we combine it with adjustments of the data structures. Basically, a free hand movement of a label causes a customized label position with zero cost to be created and inserted into the conflict graph. The solution I is also updated to have this candidate position set as the choice for the placement of the label. Note that future executions of a labeling method considers the candidate customized position with high preference.

Another point to be noted is that the user can increase or decrease the font size of a group of selected labels simultaneously. This facility combined with executions of labeling methods allows an interactive search for a solution to the Maximum Size Labeling problem [143], which is also NP-hard.

7.4.6 Selection Extension

In the previous section we showed that the user could solve labeling problems by performing some adjustments of the data structures, and then focusing the optimization methods on a region containing the modified elements. We, however, did not mention anything about the selection itself. We know that such selection needs to include the modified features, and possibly some nearby features must be relabeled to allow a low cost solution. The problem here is to decide which nearby features should also be selected. In this section we introduce a mechanism that helps the user to select related features. The mechanism computes the set of all features that may need to be relabeled in order to solve a local problem.

We present two new concepts in Map Labeling:

Definition: A feature f *affects* a feature q, represented by $f \rightarrow q$, $f, q \in F$, if there is an edge (u, v) in the conflict graph, where u is a vertex associated with any candidate position of f, and v a vertex associated with the current candidate position in use for q.

Informally, f affects q if there is a labeling solution for f that conflicts with the current labeling of q. This concept is not symmetric, as $f \rightarrow q$ does not imply $q \rightarrow f$.

Definition: Let $A(f) = \{q \in F : f \rightarrow q\}$. The *affected closure* $C(f)$ for a feature $f \in F$ is a set of related features defined recursively as follows:

$$C(f) = \{f\} \cup \bigcup_{q \in A(f)} C(q).$$

The affected closure defines all features that need to be taken into consideration when solving a labeling problem that involves a given feature f. Figure 7.14 illustrates this concept. The affected closure of f_1 is $\{f_1, f_2, f_3\}$. This means that for labeling f_1 optimally we may have to change the labeling of f_2 or f_3. In fact, the conflict between f_1 and f_3 can be solved by relabeling either of these two features. Moreover, if the label of f_1 is placed on the bottom-left candidate position of this feature, then a new overlap with the label of f_2 is created and f_2 has to be relabeled.

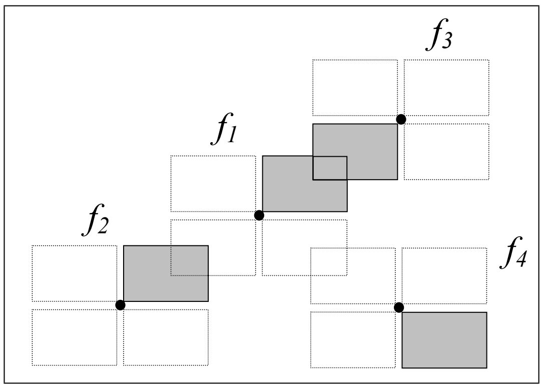


Figure 7.14: Examples of affected closures for four features. The affected closures are $C(f_1) = \{f_1, f_2, f_3\}$, $C(f_2) = \{f_2\}$, $C(f_3) = \{f_1, f_2, f_3\}$ and $C(f_4) = \{f_4\}$.

Since the affected closure is recursive, it may contain features whose candidate positions are not overlapping, but are affected indirectly. For example, $f_2 \in C(f_3)$ in Figure 7.14, even though f_2 and f_3 do not overlap. The affected closure of f_3 contains f_2 because $f_3 \to f_1$ and $f_1 \to f_2$. This is meaningful, because the overlap involving feature f_3 can be solved by keeping this feature as it is, and relabeling f_1 and possibly f_2 as described above.

We note that the affected closure is also not symmetric. For example, $C(f_2)$ does not include f_1.

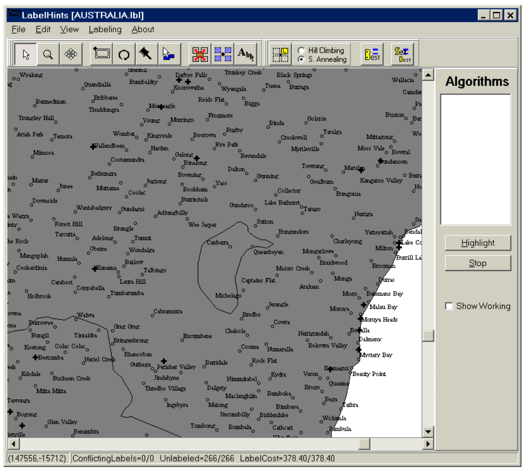
Furthermore, the affected closure for a feature f is different from the set of vertices determined by the maximal connected subgraph that contains f. The conflict graph for the labeling problem in Figure 7.14, for instance, is connected and includes all four features. However, the affected closure $C(f_1)$ does not contain f_4. Some labeling approaches [192] identify all disjoint connected components of the conflict graph, and compute a labeling individually for each connected subgraph. The advantage of taking the affected closure of a feature instead of the maximal connected subgraph is that we can reduce the size of the problem. Note, however, that this is dependent on a particular group of features that are considered a reference. In addition, it depends on the current labeling of all features. If f_4 were labeled using the top-left candidate position, then it would also be included in the affected closure of f_1.

We use the affected closure in our framework as part of an interactive tool for extending the user selection. The user selects a set $N \subseteq F$ of important features that present a problem (this can be, for example, two features whose labels overlap). The user then calls a selection extension procedure that selects all features in the set $\mathit{CLOSURE}(N) = \bigcup_{f \in N} C(f)$.

The set $\mathit{CLOSURE}(N)$ can be computed in linear time (on the size of the conflict subgraph induced by the features in this set) using a depth-first search.

7.5 The LabelHints System

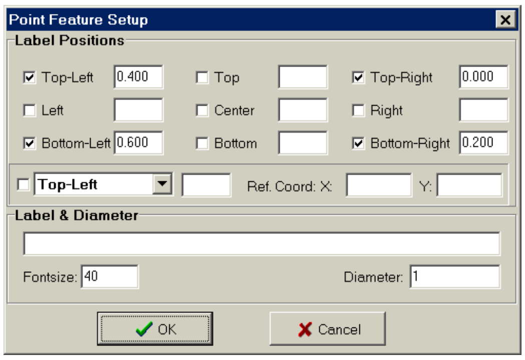
We have implemented a prototype system, called **LabelHints**, for experimenting with our map labeling approach. See a snapshot of the system in Figure 7.15.



(147556,-15712) ConflictingLabels=0/0 Unlabeled=266/266 LabelCost=378.40/378.40

Figure 7.15: LabelHints – an interactive Map Labeling system.

At the beginning of the labeling process, the system creates a trivial working solution by producing a set I with an unlabeled vertex for every feature. This is shown in the visualization by drawing crosses representing unlabeled features. The user then selects a group of features and defines candidate positions for them. Figure 7.16 shows a dialog window for setting feature attributes. After defining candidate positions, the user may call an optimization method for improving the labeling.



Label Positions

[✓] Top-Left	0.400	[ ] Top		[✓] Top-Right	0.000
[ ] Left		[ ] Center		[ ] Right	
[✓] Bottom-Left	0.600	[ ] Bottom		[✓] Bottom-Right	0.200

[ ] Top-Left Ref. Coord: X: Y:

Label & Diameter

Fontsize: 40 Diameter: 1

Figure 7.16: Feature Setup window in the LabelHints system.

Two optimization methods were implemented in the system: a Hill Climbing algorithm and a Simulated Annealing algorithm. A multi-thread architecture is supported so that the algorithm can run as an independent thread, and multiple threads can be executed simultaneously.

The user starts an optimization method by clicking on a button of the interface. The system then computes a simplified conflict graph containing the selected features, as explained in Section 7.4.3, and initializes a thread for the labeling method. The thread inputs the simplified conflict graph and produces two labeling solutions for it, which are kept internally: a working labeling and the best labeling found so far (note that these solutions are different from the working and best solutions kept by the system for the entire problem). The thread then runs until a stop condition defined by the optimization method is satisfied. During this execution, the system queries the thread in regular time intervals for a solution and integrates it with the global working solution. Either the internal working labeling or the internal best labeling can be queried; the choice is determined by the user. However, when the thread completes its execution, only the internal best labeling is available.

The user can start several threads simultaneously for the same set of selected features or for different sets. Each thread has its own simplified conflict graph and produces independent solutions. When more than one thread is available, the system queries each thread, and updates the global working solution sequentially with the results. Note that no lock mechanism of the global solution exists. This means that the result produced by a thread may overwrite the labeling computed by another thread, if they share some common features. Therefore, the user is responsible for deciding how and when to run the algorithms. Nevertheless, superposition of results is not a very serious problem as the best solution agent is implemented. The agent checks the working solution after every update with the result of a thread, and saves the best global solution.

The system provides the two visualizations described in Section 7.4.4. The visualizations are interactive, allowing the user to perform feature selection, to set labeling constraints and to perform manual replacement of the labels via direct manipulation. Parameters of the quality of the working solution and the best solution are shown in a status bar.

More information about the LabelHints system is presented in Appendix B.

7.6 Evaluation

In this section we describe an evaluation of **LabelHints** system performed with experts. The evaluation is rather informal, because a controlled experiment with human subjects (of the kind described in Sections 5.5 and 6.5) is difficult for Map Labeling. The reason for this is that Cartography is a professional activity which requires years of training and experience. One cannot perform human computer interaction experiments in this domain using untrained subjects. Further, it is impossible to assemble enough professional cartographers to conduct a controlled experiment with statistically significant results. Thus, we performed a study along the lines of "heuristic evaluation" [144] by demonstrating the system to domain experts and noting their feedback. The main difference from a traditional heuristic evaluation approach is that some experts had to work in groups (due to their locality and tight work schedules), and we aggregated the results of the different groups ourselves afterwards.

Meetings were held with cartographers from three organizations: UBD – Universal Press Pty Ltd, Sydway Publishing Pty, and The Defence Imagery & Geospatial Organisation (DIGO). UBD is a brand of Universal Press, Australia's largest publisher of mapping and travel related products. UBD publications include street directories, maps and atlases for the whole country. Sydway is part of a group of companies called Ausway, another major mapping organization in Australia that produces street directories for Sydney, Melbourne and Central Coast. DIGO is the lead agency in the Department of Defence responsible for the acquisition, production and distribution of imagery and geospatial based intelligence and data in support to the Australian Defence Force and Government decision makers. DIGO provides a wide range of services from hardcopy maps to digital products for incorporation into Geographic Information Systems.

Our first visit to UBD was on the 21st of February in 2002, when a preliminary version of the system was presented to Mr. Graham Russell, the Technology Manager, and to his colleagues. A second visit was made on the 15th of October 2002 for demonstrating the complete prototype.

We had meetings with Mr. Murray Godfrey, of Sydway, on the 12th of July, and with Mr. David

Godfrey, Director of Ausway, on the 29th of October 2002.

On the 10th of February 2003, we visited DIGO, in Bendigo – Victoria, and met with a group of cartographers under the supervision of Mr. Brian McLachlan, chief of the Geospatial Intelligence Branch.

All three companies make extensive use of computer technology to produce maps, and their existing software could place labels. UBD has an Autocad system extended in house to support Cartographic processes. This system computes an initial placement for each label by performing a simple graphical search nearby its feature for a position that implies less overlaps. Sydway and DIGO use Maplex and ArcGIS[6], respectively; these two systems implement good labeling algorithms, and allow the users to specify preferences for label placement. None of these three systems, however, support an interactive labeling optimization such that of our framework, based on focus and on a dynamic conflict graph.

Each meeting with the experts lasted between two to three hours. This involved a presentation of the interactive framework, followed by a demonstration of the **LabelHints** system and by discussions about the interaction facilities. The interactive labeling process was demonstrated for a map of iron occurrences in Tasmania (Figure 7.17), a map with cities in Iowa-USA (Figure 7.18), and a randomly generated map with proper names (Figure 7.19). The experts were asked about the effectiveness of using our interaction facilities in comparison with their normal way of performing map labeling tasks. Even though our framework supports fully automatic map labeling, we were more interested in using virtual overlaps and labeling constraints for solving labeling problems, rather than by manual changes. The experts were also questioned about the system in general and about their understanding of the conflict graph concept. Our meetings with Sydway's experts were at the Information Visualisation Laboratory of the University of Sydney, where they could use a special table that we built for interactive optimization. Details about the table are given in Appendix A.

The feedback on the system was gathered informally via comments from the experts. The salient points were as follows:

- The cartographers at UBD mentioned that they spend a significant amount of time manually improving map labeling solutions. However, they showed that their maps have a high degree of information, with many lines and area features that need careful and detailed refinement. The maps are used many times and are very rarely modified after being finished. Therefore, a

[6]Maplex and ArcGIS are products of the Environmental Systems Research Institute, in California, USA. For more details about these products see *http://www.esri.com/software/*.

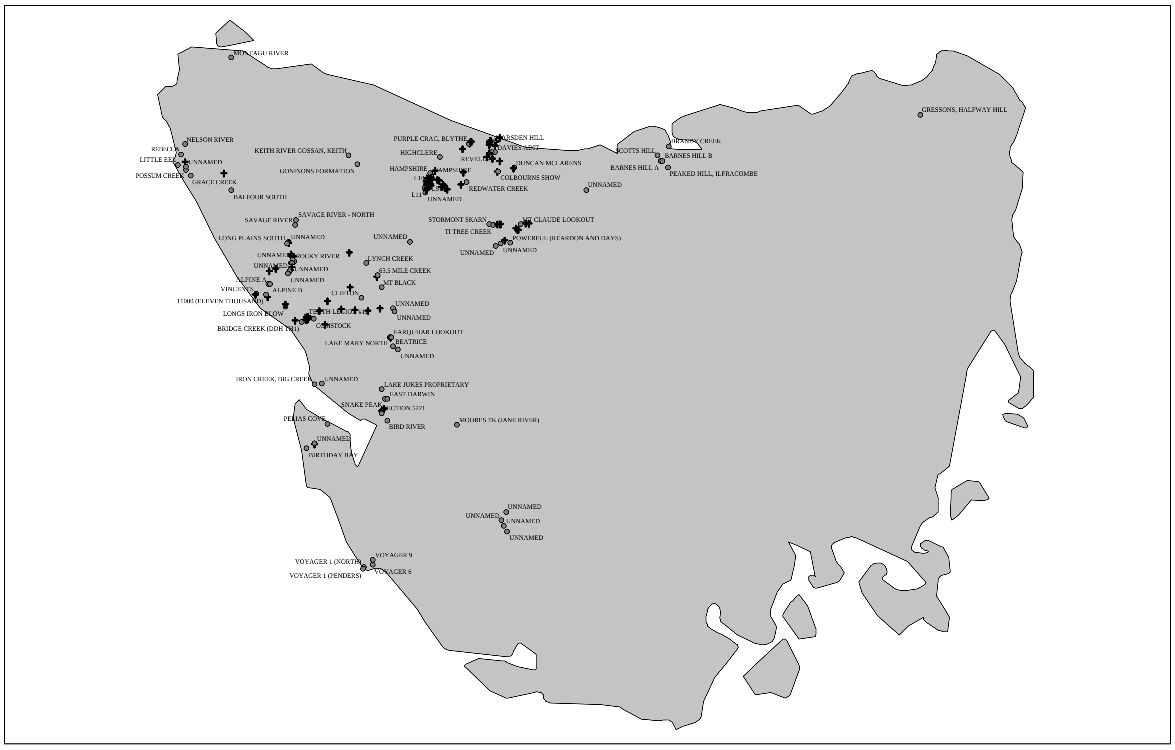


Figure 7.17: Map of Iron (Fe) occurrences in Tasmania. The labels consist of names of several localities, including towns, mines and deposits. Some places are labeled "UNNAMED". This incorporates data which is: (c) Commonwealth of Australia (Geoscience Australia, *http://www.auslig.gov.au*) 1990.

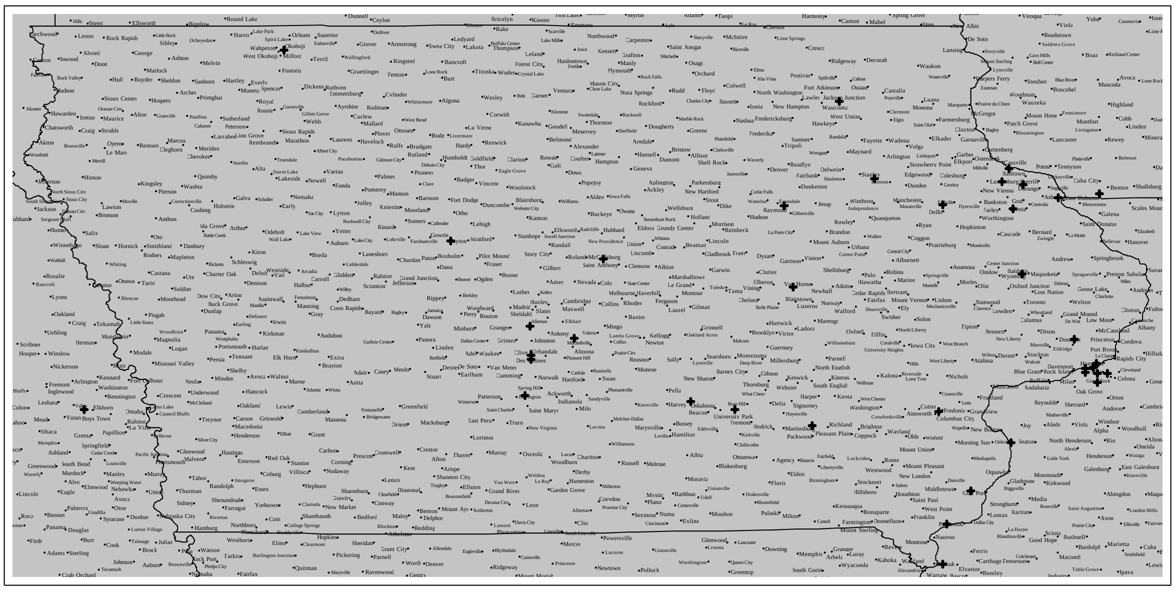


Figure 7.18: A map of towns in Iowa, USA. Data from the National Mapping Information Server of the United States Geological Survey (*http://mapping.usgs.gov/*).

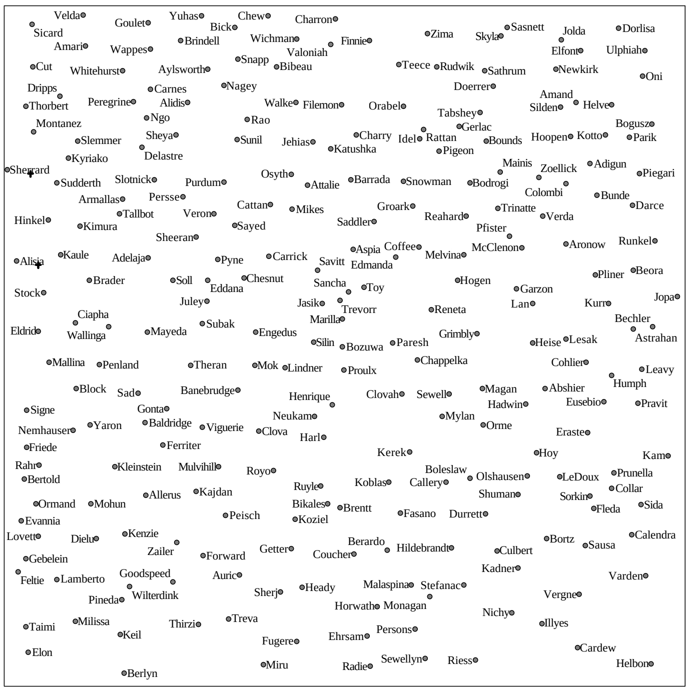


Figure 7.19: A randomly generated map with proper names. The proper names are from the SCOWL collection, the Spell Checker Oriented Word Lists (*http://wordlist.sourceforge.net*).

laborious manual refinement is justifiable. The experts noted that our interactive framework may be useful for Geographic Information Systems, where point features are more common, and where changes of the geographic information stored in databases happen more often.

- Sydway's cartographers also mentioned that they spend a considerable amount of time in manual refinements of detailed labeling solutions. Almost every label needs to be repositioned after an initial automatic placement. The experts commented that this labeling adjustment is much easier to perform via a fully manual drag-and-drop operation of the labels, than by using some advanced interactive facilities such as the virtual overlap. The explanation for this was that the users could effectively place the labels at the precise positions that they wanted by a free manual placement. Figure 7.20 shows the map of the Sydney Centre that appears in Sydway Street Directory; this is a very dense area that needed careful refinement.

- The comments from the cartographers at DIGO regarding labeling improvement were that moving labels manually seems more effective for small areas, while the interactive tools can be useful for dense areas with several labeling problems.

- The cartographers of Sydway and DIGO pointed out two important resources that were not

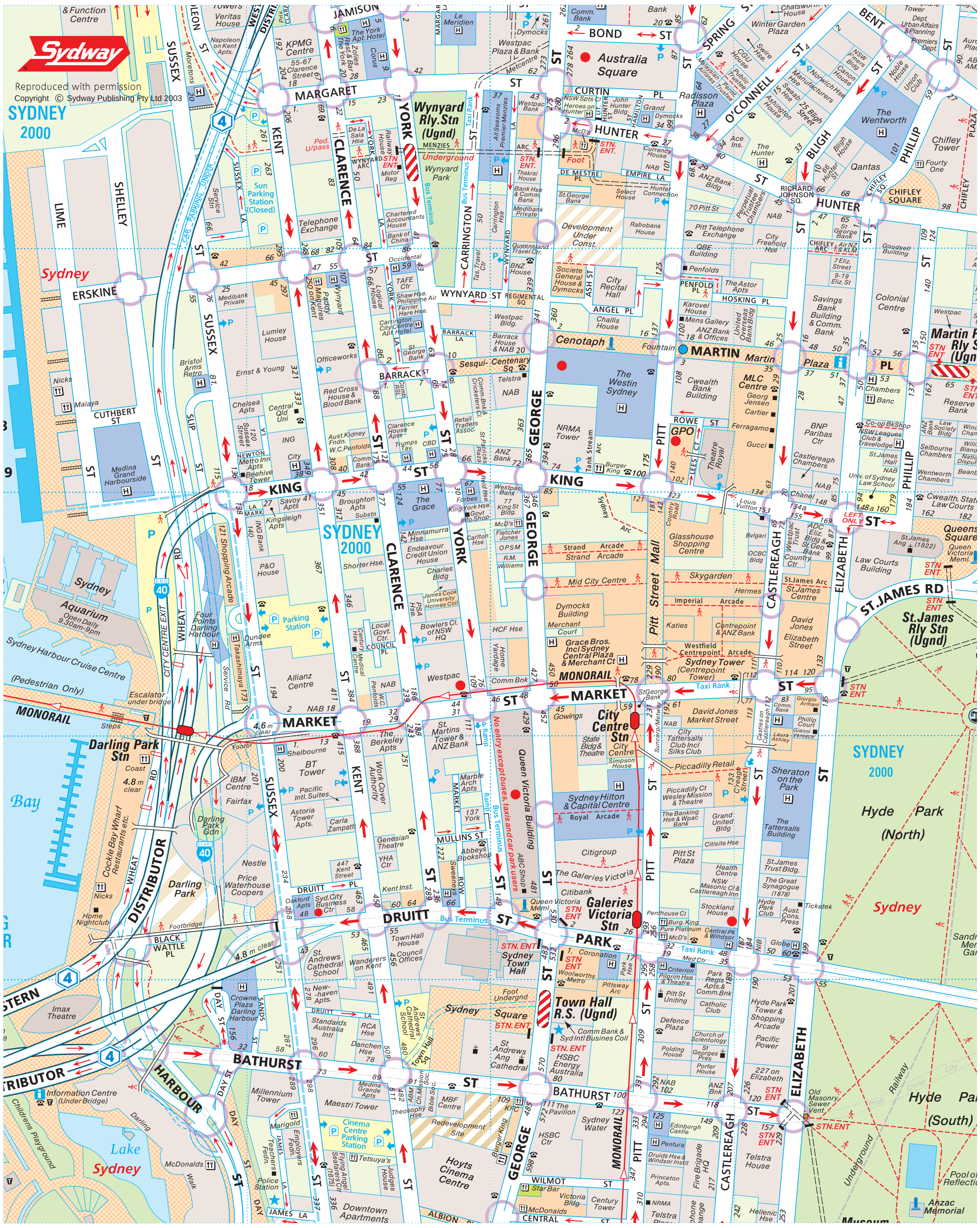


Figure 7.20: The Sydney Centre. Map provided by Sydway Street Directory of Greater Sydney. Copyright Sydway Publishing Pty. Ltd. 2003.

fully implemented in the system: weights assigned to point features (indicating the importance of displaying their labels), and levels of font size for different features. Font size and feature weight usually depend on population size, when the point features represent localities. Even though our system supports labels with different font sizes, we do not relate this resource directly to feature importance in the labeling process. In addition, feature importance only exists partially in our system, and it is represented by the labeling constraints.

- DIGO's experts proposed several extensions to the system. They mentioned that identifying ambiguity and other labeling problems in a large map can be very time consuming. They then suggested that the system could highlight pairs of labels that may be problematic (based on some heuristic checking), and ask the user about running an automated improvement process. If the user answers 'yes', then the system could add a virtual overlap between the problematic labels, select some affected features and run an optimization method. Some final suggestions obtained in DIGO included: extending the system to handle layers of features, as well as overlaps between elements in different layers; implementing a constraint enforcing a minimum offset between labels and between labels and features; and automatically hiding some labels of less importance when zooming out.

We noted that suggestions for extending the system dominated the comments of the domain experts. They proposed not only traditional cartographic tools and ideas, but also semi-automatic routines that might help the map labeling optimization. Some suggestions show promising directions for automating the interaction facilities[7]. In general, the experts expressed that the framework has a good potential for helping cartographers in real map labeling applications, but that more features need to be added to the system.

The experts' comments indicate that fully manual postprocessing can be more effective than the human-computer collaborative approach for high quality maps. The disadvantages of some interaction facilities are that they may not satisfy entirely the experts' desires, and can also create new labeling problems. Figure 7.21 illustrates this problem. An overlap is added by the user to express the existence of ambiguity between the label "Webster City" and the feature labeled "Blairsburg". The intention is to force "Webster City" to be moved to a candidate position that is distant from "Blairsburg". Such an operation can be performed by clicking with a particular tool on the two ambiguous elements, selecting some features for reorganization, and calling an

[7] More ideas for semi-automatic facilities emerged from the discussions with Sydway's cartographers, and are presented in the Remarks Section of this chapter.

optimization method. The optimization method, however, finds a low cost solution that consists of setting “Blairsburg” as unlabeled, which is a worse solution. In other examples, the method may relabel the problematic features properly, but can create new ambiguity between other pair of labels. For cases like this, a simple manual adjustment where the experts move labels to the positions that they want is still more effective.

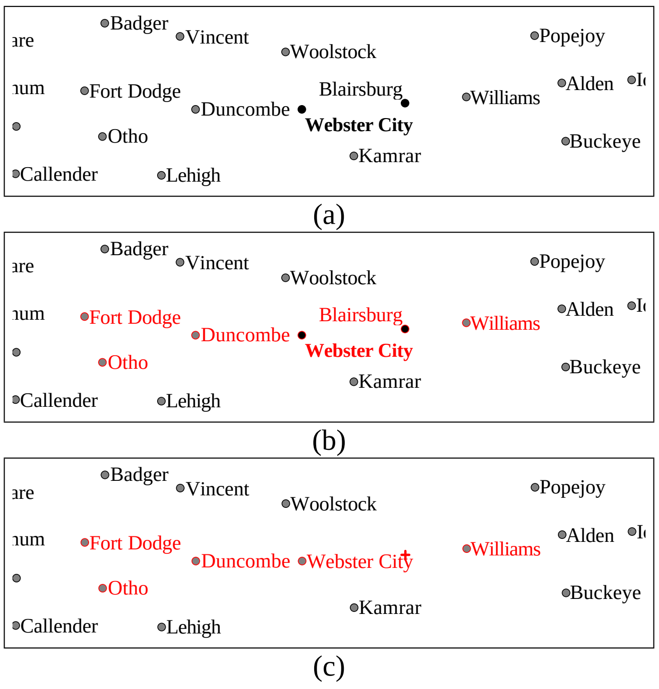


Figure 7.21: An optimization task where a new problem is created by using virtual overlaps. The pictures show a sequence of steps executed by the user for relabeling a section of a map: a virtual overlap is defined between the label “Webster City” and the feature “Blairsburg” (a), some nearby features are selected (b), and the Simulated Annealing algorithm is called for computing a new labeling (c).

The benefit of using virtual overlaps was recognized for cases where several labels have to be moved. Figure 7.22 shows an example in which there are many labeling problems and the user defines overlaps and constraints for them. By calling an optimization method the user obtains a new solution where most of the problems were solved. The method relabeled the problematic features, and reorganized some nearby labels. This solution can be fined tuned by manual adjustment. One domain expert at DIGO noted that an undo facility is really critical here, to allow the user to revert to the initial solution in case the computer-generated labeling is visually worse (although it can be better in terms of the quality function).

We realized that the selection extension tool is important for helping the user to focus the optimization methods. The user’s guesses about which features are affected by another feature can be

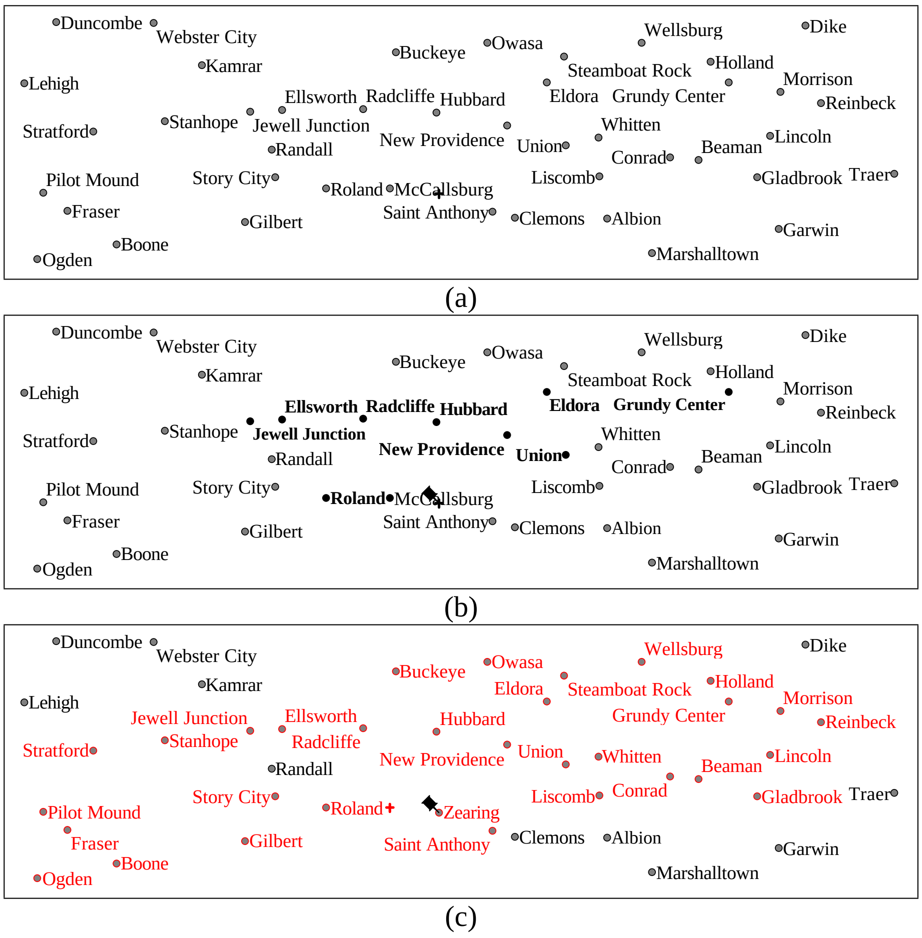


Figure 7.22: An optimization task where several labeling problems need to be solved. The figure shows: (a) the initial labeling with aesthetic problems and with a feature, “Zearing”, that needs to be labeled; (b) the visualization after adding virtual overlaps and a labeling constraint; and (c) a new labeling solution produced by the focused Simulated Annealing.

wrong. Sometimes nearby features on the map are not related at all, and do not have to be included in the selection. The opposite situation, where distant features seem unrelated but are affected, also occurs. Nevertheless, the user may decide to choose even a smaller subset of the extended selection, particularly when the image is scaled up and the selection runs out of the screen area. Figure 7.23 shows the extended selection computed for a feature on the center of the screen. The user may reduce the selection here, in order to avoid changes in areas that currently are not visible. Note, however, that this can reduce or eliminate the chances of finding an optimal labeling for the problem.

A useful resource is the possibility of having two different algorithms. We noted that Simulated Annealing is difficult to tune perfectly to all problems. We have set its parameters for a slow schedule, but it may stop before some easy improvements are completed. Running the Hill Climbing

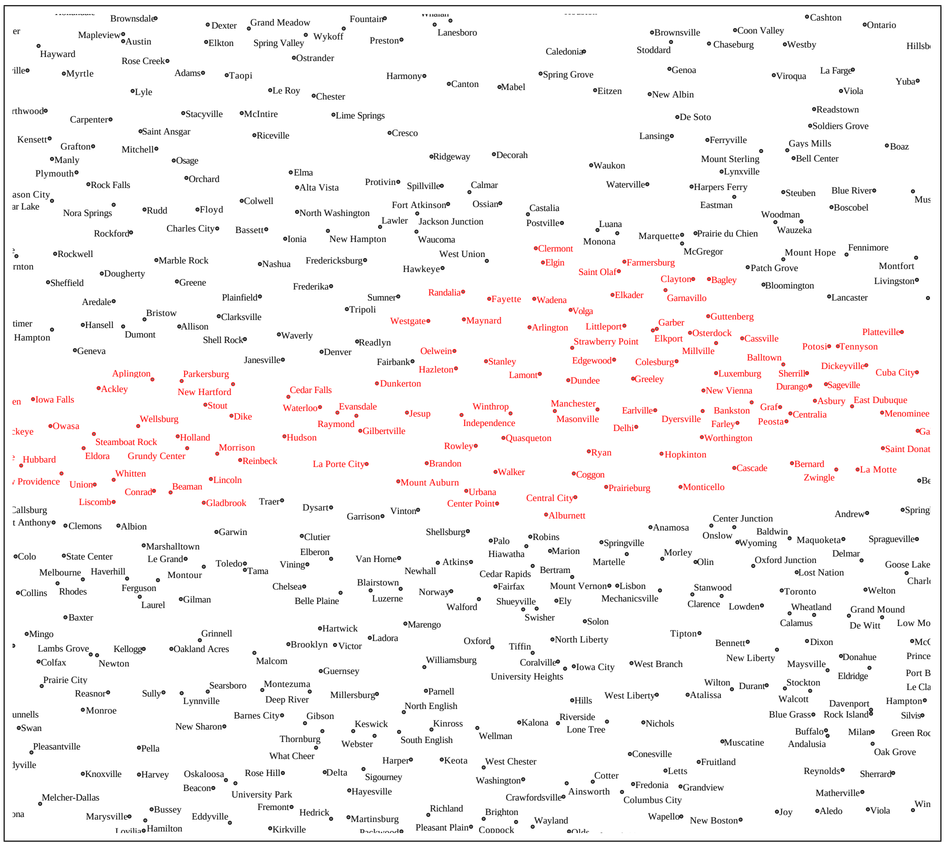


Figure 7.23: Feature selection running out of the screen area (visible when printed in color).

algorithm after Simulated Annealing, or as an alternative method, could achieve better solutions.

Another interesting resource is the multi-thread mechanism in the LabelHints system. As we mentioned in Chapter 2, Simulated Annealing is a stochastic method that can yield different solutions in new executions. We allow the user to exploit this factor by starting several instances of this algorithm on a selected labeling problem. Better results can be obtained with this approach (the best labeling is automatically filtered by the best solution agent). The user can also perform changes while the algorithms are running, and the system can take advantage of multi-processor machines.

A final point about the evaluation is that the visualization of the conflict graph did not appear to be useful for the experts. The cartographers commented that they understood the concept of the conflict graph and the basic operations performed on it by the interaction facilities. However, the visualization itself can be confusing for dense regions of the map, and does not seem to provide any new information to the user. Refreshes of the drawing of the conflict graph (which happen when the working solution or the problem changes) also demand considerable computational resources, and this decreases the interactivity of the system.

7.7 Remarks

7.7.1 Expert Heuristics

The cartographer from Sydway described simple strategies that they employ for solving overlap problems in dense areas. These are:

1. Shifting the labels slightly so that an overlap is resolved. This is a manual placement where the labels are shifted horizontally or vertically to another place, while still touching their features.
2. Reducing the font size or setting other font attributes such as character spacing.
3. Breaking the label into two or more lines.
4. Abbreviating names.

There is a strong potential for exploiting these strategies in a semi-automatic approach. The idea is to implement heuristics that incorporate the strategies, and allow the experts to control their execution as a normal optimization method. The experts can decide when to execute the heuristics and on which part of the problem to focus.

The first strategy, for instance, suggests using the Slider model. An algorithm for the Slider model can be employed such as the one presented by Hirsch [91], in order to improve the labeling of a few selected features that are overlapping or are too close to each other. A mapping between the models can be implemented: non-selected labels in the framework are treated as obstacles in the Slider model, and the final results are converted back into the Fixed-Position model as customized candidate positions. It is also possible to use the Fixed-Position model to simulate the Slider model, by treating the area around each feature as a discrete space, and create a candidate position for every possible place near the features. The costs for the candidate positions can be determined by interpolating the costs associated to the original 4 or 8 predefined-candidate positions. This is feasible only if the number of selected features is small. After the computation of a labeling solution, the recently-created candidate positions are destroyed, except for the ones that are chosen for the solution.

Similar ideas can be implemented for the other strategies, and they are only used when necessary. Note that this does not prevent incorporating such ideas in fully automatic processes. In fact, the last two strategies (breaking labels in lines and using abbreviations) are already implemented in an automatic algorithm in Maplex. However, it may be interesting to define labeling rules that can be easily customized and reapplied by the user on some local regions of the map.

7.7.2 The Conflict Graph

The conflict graph represents the main data structure used in the framework. Therefore, some efficiencies are needed to compute and maintain it, and to reduce memory usage. Basically, it is necessary to use efficient algorithms from computational geometry for searching for candidate positions that are nearby to or overlap with a given feature or label position. This is for creating edges in the conflict graph. The choice of the representation for the conflict graph is also important, so that not all edges have to be defined explicitly. We use the data structure described by Christensen *et al.* [33], which keeps a list of candidate positions for every feature. Thus, the clique between these positions is implicit. Further, when saving a final solution onto disk, it is sufficient to record only the positions in use and the ones defined by the user. The remaining positions can be reconstructed when loading the graph.

Note that the conflict graph has its limitations, which are related to the fact that it describes a simple binary relation between label positions. The conflict graph cannot be used to exclude a particular placement involving three or more label positions without also excluding partial configurations of these positions.

7.7.3 Comparison with the Traditional Labeling Approach

The main difference between the traditional map labeling framework supported in commercial packages, such as Arcview and Maplex, and the User Hints framework lies fundamentally in the alternatives offered to the user to improve a labeling solution. In those packages the user can only interact at the beginning and at the end of the optimization process. At the beginning, the user defines a set of placement rules for each class of features or for the whole map. These rules are then used by an optimization method to create a labeling placement. After the placement, the user can revise the solution; however this can only be done in one of the following ways:

- through a fully manual post-processing adjustment; or
- by modifying the rules and recomputing the labeling solution entirely – the previous labeling placement is discarded.

The traditional labeling approach does not allow the user to refine the problem and to focus an optimization method to improve a particular region of the map. Moreover, re-executing the method causes improvements manually performed by the users to be lost. Consequently, the traditional approach cannot be used in a continuous and incremental way.

7.7.4 Extensions to the Framework

Although we restricted our study to the problem of labeling point-features with rectangular labels, the standard combinatorial model used in our framework is easily extensible to handle any type of feature and label shapes, as shown by Edmondson *et al.* [61]. Studies that consider weights for map features may also be incorporated [149].

Another point to note is that traditional labeling systems treat ambiguity by allowing the users to specify a minimal distance between two labels or between labels and features. However, this is done uniformly to all labels, and can prevent solutions that have close labels, but are still acceptable. Thus, a possibility is to support a flexible offset constraint, where the expert can specify different offset distances for distinct regions of the map. Such a facility can be combined with the selection mechanism: the user selects a group of features, and sets the offset constraint. The system then creates a virtual overlap for every pair that violates the constraint. In contrast, relaxing the offset would cause some virtual overlaps to be deleted.

Since the framework already implements interaction facilities for adding and removing edges from the conflict graph, the implementation of an interactive offset constraint seems straightforward.

CHAPTER 8

General Remarks

This chapter discusses the User Hints framework in light of experience with the systems described in Chapters 4, 5, 6 and 7. We start in Section 8.1 with a discussion of the implementation of focus and user-defined constraints in optimization methods. We then present in Section 8.2 guidelines to apply the User Hints framework to other optimization problems. A justification for investing in the development of interactive optimization is presented in Section 8.3. Next, in Section 8.4, we suggest ideas for using our framework as an environment for helping the development of new automatic methods. Finally, we conclude in Section 8.5 with possible extensions to the framework.

8.1 Constraints and Focus: Implementation Issues

The optimization methods presented in the previous chapters were modified to support user-defined constraints and focus. We now discuss some issues that arise when implementing such modifications, and present general directions for this task. We restrict our attention to two major categories of optimization methods: problem-dependent heuristics (such as the heuristics for the Sugiyama method described in Chapter 5), and meta-heuristics (such as Hill Climbing, Simulated Annealing, and Genetic Algorithms used in Chapters 4, 6, and 7).

8.1.1 Constraints

We found that user constraints were much more difficult to encode than focus. Moreover, constraint implementation was harder for problem-dependent heuristics than for meta-heuristic methods.

For the case of problem-dependent heuristics, the implementation was based on a careful study of an existing code; this varied from algorithm to algorithm and was time consuming. For meta-heuristics, the implementation of user constraints was more straightforward, as it could be done by modifying an objective function or including constraint-satisfaction subroutines.

Despite the differences in implementation, we identified some common options for extending an optimization method in order to support user-defined constraints:

(a) Modifying the algorithm, so that a solution changes in a way that never violates a constraint. In other words – using the definitions presented in Section 2.1.2 – the algorithm only applies elementary operations on a solution S that moves S to (or maintains S in) the feasible domain. This alternative was adopted for the implementation of the MOVEX routine in the Genetic Algorithm in Chapter 6, in order to prevent vertex overlap.

(b) Allowing infeasible solutions to be created, and including a cost factor for constraint violation into the objective function with a high importance. In this case, the optimization method naturally searches for a feasible solution. Such an approach was broadly used in this thesis, for example, in the algorithms for Graph Clustering in Chapter 4, and for treating layout constraints in the Genetic Algorithm in Chapter 6.

(c) Allowing constraints to be temporarily violated, but implementing constraint-satisfaction routines that can be called afterwards to adjust the solution and guarantee feasibility. Examples of such routines are the *FixConstraint* algorithm presented in Section 5.3.2, and the *SOLVER* operator described in Section 6.3.4.

We believe that the best approach was to use a combination of the options presented above, instead of a single alternative.

8.1.2 Focus

Focus can be implemented by adding an attribute *selected* to every variable of the problem, which indicates whether it is selected or not. The optimization methods are allowed to read all variables, but can only change the value of the selected variables. In most cases, this condition can be guaranteed by simply checking the attribute *selected*, before trying to change a value. The *Barycenter* algorithm in Section 5.3.2, for example, implements this simple approach. In other cases, such as for the Layering Assignment algorithm in Section 5.3.2, some extra processing needs to be done to guarantee that constraints and desirable properties of the solution are satisfied. This usually happens when two variables, say p and q, take part in a constraint c. An optimization method may change p, and then try to change q to maintain the satisfaction of c (considering that c was satisfied previously, but is now violated by the new value of p). However, if q is not selected, then it cannot be changed; thus, the method has to reset p to its original value, or readjust it to another value in order to guarantee feasibility.

The implementation of focus can go even further by modifying the optimization methods to concentrate action only on the selected variables of the problem. Since non-selected variables are never changed, the methods do not have to consider them until it is necessary to read their values. This is exactly the type of modification that allows the use of focus to reduce the exploration of the solution space. Almost all algorithms in this thesis used this approach. The Barycenter algorithm, as an example again, skips iterations of its internal loops that involve layers containing only non-selected vertices (although this is not shown explicitly in the code in Figure 5.7 on page 80). For the Genetic Algorithm, we extract the elements that are necessary to reconstruct the selected part of a drawing, and perform operations only on them; moreover, we avoid recomputing the quality of non-selected areas of the drawing.

These two kinds of modification, preventing changes of the value of non-selected variables and concentrating action only on the selected variables, are the main characteristics of the focus mechanism.

Note that the selected attribute can be assigned individually to each variable of the problem, or to a group of variables. For example, in the Graph Drawing case study in Chapter 5, the selection attribute is associated with a vertex, which contains two variables: the X and the Y coordinates of its position on the grid. If the vertex is selected, then either X or Y or both variables can be changed by an optimization method. We could instead allow a more fine-grained selection, where the user selects the variables X and the Y independently. In that case, the visualization should also be redesigned to represent partial or complete selection of the coordinates of the vertices.

Supporting Focus by Creating a Simplified Problem

The preprocessing approach used in the Map Labeling case study, which creates a simplified labeling problem (see page 168), represents a more natural strategy for implementing focus. The approach is based on the following steps:

1. Creating a simplified problem instance that contains only the necessary variables to be investigated. These variables are the selected variables, as well as the non-selected variables that are dependent on the selected ones. Informally, we consider that two variables are dependent

if they are involved in a same constraint, or if they are tightly coupled in the objective function of the optimization problem – so that the problem cannot be properly solved by setting these variables separately and combining their values afterwards.

2. Possibly fixing the values of all non-selected variables, so that the optimization methods do not have to be modified to distinguish between selected and non-selected elements.
3. Producing a solution for the simplified problem by applying an optimization method.
4. Integrating the solution for the simplified problem with an existing solution for the complete problem.

The main idea is to modify the instance of the optimization problem, rather than the optimization method itself. This can be achieved by discarding all variables and constraints that are not affected by the selected variables. As a consequence, the problem becomes simpler and can be solved with less computational effort. Moreover, if it is possible to set the non-selected variables as constants, then traditional optimization methods can be used without modification.

The preprocessing approach was effectively used for Map Labeling, and can be applied to other optimization problems. Consider, for example, a version of the Graph Clustering problem which aim is to reorganize an existing clustering solution $S = (N_1, N_2, \ldots, N_k)$, where $N_1, N_2, \ldots, N_k$ are clusters of the vertices of a graph $G = (V, E)$, so that the number of intercluster edges is minimized and the sizes of the original clusters are preserved. Consider also that an algorithm A exists for this task, and that $A(S, G)$ tries to improve S as much as possible by swapping pairs of vertices in different clusters. Suppose now that the user wants to focus on just two selected clusters N_1 and N_2 to be reorganized; in this case a simple call to $A(S, G)$ may not work properly, as the algorithm may modify any cluster in S. We can, of course, adjust A to recognize and preserve non-selected clusters, such as $N_3, \ldots, N_k$. However, a more interesting approach would be to compute a *simplified* clustering solution $S' = (N'_1, N'_2)$ and a *simplified* graph $G' = (V', E')$, where $N'_1 = N_1$, $N'_2 = N_2$, $V' = N_1 \bigcup N_2$ and $E' = \{(u, v) \in E : u, v \in (N_1 \bigcup N_2)\}$. We could then execute $A(S', G')$ to improve S', and update the solution S by replacing N_1 with the improved partition N'_1, and N_2 with N'_2. This approach demands additional processing time for creating S' and for updating S; nevertheless, it offers a modular way of implementing focus, and can be easily integrated with any clustering algorithm (given that it does not change the size of the existing clusters).

Unfortunately, the preprocessing approach does not work for all problems. The Directed Graph Drawing problem studied in Chapters 5 and 6 is an example. We cannot create a subgraph containing few vertices, run a graph drawing algorithm on it, and expect the resulting drawing to be integrated well with an existing drawing of the entire graph. As we discussed in Chapter 6, the selected and non-selected vertices of a drawing are strongly interdependent. Consequently, the non-selected vertices always have to be taken into consideration. Note that the genetic algorithm implemented in Chapter 6 maintains an internal copy of the whole drawing in order to query the position of the non-selected vertices and edges.

In general terms, given an optimization problem with a set of selected variables and a set of unselected variables, we can identify three *dependency conditions*:

1. *Independent selection* – the selected variables are independent of the non-selected ones. As a consequence, the preprocessing approach can be applied, and the optimization methods do not have to be modified to support focus. An example is the version of the Graph Clustering problem described in this section.

2. *Proportional-dependent selection* – there is dependency between selected and non-selected variables, but the number of dependent non-selected variables is small or proportional to the number of selected variables. In this case, the preprocessing may still be used, but the dependent non-selected elements have to be incorporated into the simplified problem. This is the case of the preprocessing approach for Map Labeling.

3. *Strongly-dependent selection* – all non-selected variables are involved in a dependency relationship with at least one selected element. For example, in the Directed Graph Drawing problem, every selected vertex of the graph is dependent on all non-selected vertices, since the latter ones are necessary to compute the number of edge crossings and to avoid vertex overlap. The most appropriate approach for this case is to implement focus by modifying the optimization methods as proposed in the previous section.

The dependency conditions described above are based on the assumption that the values of all selected variables can be freely changed by an algorithm. However, if a more restrictive optimization method is used which changes only a subset of the selected variables and/or never violates some constraints, then the selection may be treated as a simpler case. For example, the strong dependence in the Graph Drawing problem (due to the fact that both X and Y coordinates of the selected vertices can be set to any position) does not have to be fully considered when using the *Barycenter* algorithm. This algorithm modifies only the X-coordinate of the vertices. Moreover, the barycenter improvement of a vertex v involves checking overlap constraints with vertices in the same layer, and recalculating certain aesthetic measures (such as the number of edge crossings) with vertices in the layers immediately above and below v. Therefore, there is a possibility of creating a simplified problem instance by using the preprocessing approach. Such a problem would include only selected vertices and the vertices in the layers immediately above and below the former ones. Non-selected vertices that are in layers far away, and that are not connected to any selected vertex by an edge or user constraint do not have to be added to the simplified problem.

Finally, the Selection Extension function presented in Section 7.4.6 also has an important effect on the implementation of the focus facility. The impact of this function can be better understood now that we have introduced some basic concepts about the dependence between selected and non-selected variables. What the Selection Extension function does is to augment the set of selected variables by adding to it the dependent non-selected variables. This process is repeated recursively until no dependent element exists in the set of non-selected variables. As a result, the sets of selected and non-selected variables become independent, and the first dependency condition (independent selection) applies. Such a function may be useful in other optimization problems to extend the set of selected variables and allow the preprocessing approach to be used.

8.2 Applying the User Hints Framework

This section presents a process consisting of six steps for applying the User Hints framework to other optimization problems (see diagram in Figure 8.1). We propose guidelines for each step based on our experience with the case studies in Chapters 4 to 7.

8.2.1 Verifying the Suitability of the Problem

The first step for using the framework is, in fact, to verify whether the optimization problem is suitable for interactive processing or not. Some factors mentioned in Chapter 1 already help to perform this test; they provide a characterization of complex problems that tend to be dependent on human intervention:

- Problems that are computationally difficult, usually involving many objectives, constraints and variables, and for which the current hardware technology cannot provide the desired

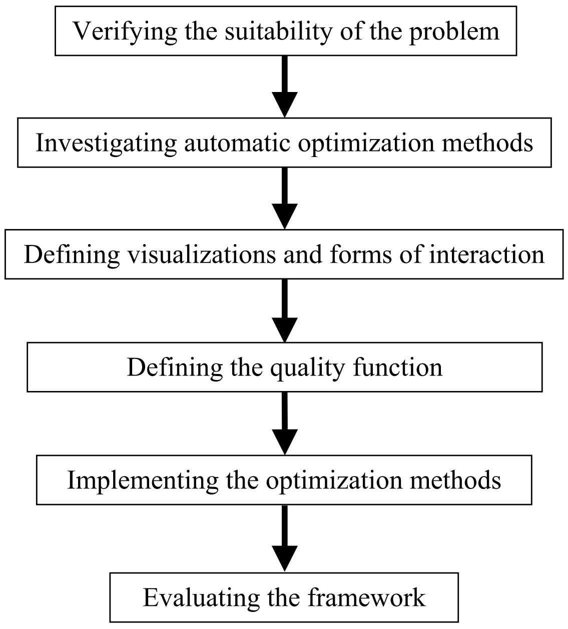

Verifying the suitability of the problem
Investigating automatic optimization methods
Defining visualizations and forms of interaction
Defining the quality function
Implementing the optimization methods
Evaluating the framework

Figure 8.1: A process for applying the User Hints framework to optimization problems.

solution in an acceptable amount of time.

- Problems for which there are equally optimal solutions according to objective functions, so that domain knowledge is necessary to distinguish between these solutions.
- Problems that are not completely known and change over time.
- Subjective problems that are difficult to specify formally.

The best test, however, is to look at the domain and identify how the problem is currently solved for practical applications. If manual processing is part of the current practice, then there is a possibility of using interactive optimization. As we saw in the previous chapters, Graph Drawing and Map Labeling are of this nature.

8.2.2 Investigating Automatic Methods

If the problem is suitable for interactive optimization, then the next step is to catalogue and study automatic methods that can be used to solve it. For each method considered, we should identify:

- Negative and Positive aspects – related to the effectiveness and efficiency of the method in solving the proposed optimization problem. Such aspects include processing time and memory usage, support for the entire set of constraints of the problem, and effectiveness in optimizing the complete objective function (or functions). This information is important because the methods usually present tradeoffs, such as being able to obtain the optimal solution, but subject to excessive computation time. In addition, many simple heuristics are suitable to restrict problems, which are simplified models of real-world problems. Knowing these aspects helps to decide which method to use and in what situation.

- Characteristics that may contribute to a better integration with an interactive environment – verifying if the method supports dynamic optimization, allows an incremental improvement of an existing solution, offers indication about its progress (such as expected time to complete the execution), and if it is flexible for change (so that focus and new types of constraints can be easily included).

The positive aspects of the methods tell us about parts of the optimization problem that can be solved automatically. On the other hand, the negative aspects that are common to all methods indicate possible areas for improvement via human interaction.

Note that the investigation of automatic methods can cause a revision of the suitability of the problem. We may find that a combination of existing methods or a new algorithm, recently published, already provides the effectiveness and efficiency that the user desires. In that case, the efforts should be redirected to implement an automatic approach. The use of an interactive framework should be postponed until new evidence shows that it is necessary. In Section 8.3, we discuss other issues related to deciding between a fully automatic or an interactive optimization.

8.2.3 Defining Visualizations and Forms of Interactions

The visualization tool must provide useful feedback, and allow the user to directly manipulate the elements of the optimization process.

Several general guidelines exist for developing effective visualizations [31, 141, 174]. In the present section we discuss some criteria more specific to the User Hints framework. Basically, all visualizations should allow a quick identification of good and bad aspects of the solutions being developed and of the progress of the optimization methods. They should also offer intuitive user interaction for performing the three main types of hints – adjustment of constraints, focus and manual change of the variables of the problem.

Examples of visualizations that can be presented to the user were described in Section 3.1.6. They include: a picture of the working solution, textual information about the quality of the working and the best solutions, and an indication of the progress of the optimization method and updates of the best solution.

The picture of the working solution is the main visualization, since it integrates most of the information to be presented to the user as well as interaction facilities. Such a picture can be designed by choosing a set of graphical objects for representing the elements of the solution and of the optimization problem. These elements are the variables, the objectives and the constraints. The graphical objects usually have attributes (such as color, position on the picture, shape, etc.) that can change to reflect the state of the elements of the optimization (the value of the variables, whether a variable is selected or not, whether group of variables contributes positively or negatively to the objective function, and whether a constraint is violated or satisfied). For example, in the Graph Clustering problem in Chapter 4 we use the size of a circle to indicate cluster size. Moreover, if a cluster violates a size constraint, then the perimeter of its associated circle is highlighted in red. In the Graph Drawing problem, edges are drawn as lines, and are painted in black if pointing downwards, or red if pointing upwards. Recall that red edges represent undesirable aspects of the solution.

The task of developing a good visualization consists then of designing meaningful graphical objects, and defining an intuitive mapping function between their attributes and the state of the elements of the optimization process. In general, this task depends on creativity, and differs from application to application. Nevertheless, good ideas for effective visualizations may come from the following sources:

- The domain of experts – quite often a graphical representation has already been developed by domain experts to visualize and solve the problem. Examples are the basic graphical standards that we used for drawing the vertices and edges in Chapters 5 and 6, and the map view for the Map Labeling problem in Chapter 7. Visualizations developed by domain experts may not be the best ones, but they are certainly a good starting point. Note that interactive systems that implement such visualizations probably look more natural and easy to use to a domain expert than software based on a new visualization. This reduces the time to learn.

- General visualization techniques – many visualization techniques exist that can be applied to different domains. One approach is the Parallel Coordinate visualization illustrated in Figure 2.4, on page 22, which can be used to represent the relationship between multidimensional data. The Graph Drawing approach is another example. There are many books about visualization techniques [31, 43, 174, 195].

Usually more than one visualization is possible for the framework; they may implement distinct graphical objects and/or different mapping functions. Using multiple visualizations is helpful when each describes the optimization process from a different perspective that enhances user comprehension. Even when only one visualization is available, it may be interesting to implement features for tuning the view, so that certain aspects of the optimization process can be temporarily emphasized or hidden.

Audio feedback and animation are also features that can be employed as part of the visualization. We use both, for example, to signal updates of the best solution. In addition, we have experimented with the idea of displaying samples of intermediate results produced by the optimization methods. This allows the user to follow incremental improvements of the working solution more easily, when the consecutive screen shots are not too different from each other.

The second role of the visualizations, in supporting direct manipulation, can be implemented by reversing the mapping function between the graphical objects and the elements of the optimization process. The aim is to allow the user to control the optimization process by manipulating the graphical objects. User Interaction can be implemented for the following tasks:

- Changing the value of the variables of the problem in the working solution.
- Inserting and deleting constraints.
- Adjusting the objectives.
- Selecting variables of the problem for focus.
- Executing the optimization methods.

Details for these interactions depend on the specific optimization problem, on the visualizations available, on general software usability guidelines, and also on creativity. We may implement only a small subset of all possible methods of interacting with the optimization process. A minimal requirement is to allow the user to improve at least the aspects that the optimization methods cannot solve by themselves. In a complete approach, the user may adjust every single element of the optimization (variable, constraint, and objective) directly. Furthermore, the form of interaction can be designed differently for different visualizations. For instance, moving a graphical object in one visualization may set the value of a variable, while in other visualization a similar move may change several variables at the same time. Naturally, principles of usability should be observed, such as maintaining consistency of the interface and providing feedback for the user's actions [144, 171].

It is important to note that new types of user constraints can be developed during the design of the visualizations. In general, thinking about a visualization offers the designer of the framework an opportunity to look at the optimization process more deeply, and identify different ways of controlling it. Constraints created at this stage can be tightly associated to the particular visualization in mind.

A final task to be executed in this step is to decide the level of flexibility of the manual change hint, in terms of constraint satisfaction. In some cases, it is useful to allow the user to violate constraints. In other situations, constraint satisfactions must be ensured by restraining the user actions or calling subroutines to solve violated constraints, as suggested in Section 8.1. In our GDHints system (Chapter 5), for example, the FixConstraint algorithm is automatically executed after a manual change in order to solve vertex overlap.

8.2.4 Defining the Quality Function

The quality function can only be defined completely after identifying the types of interactions that are intuitive for the user. This is because it depends on new types of constraints and policies regarding constraint violation that are decided in the previous step.

The quality function must include measurements for the traditional objectives and constraints of the optimization problem, and optionally for new user constraints that were identified as important. Costs for constraint violation are included so that feasible solutions (or solutions that only violate a few constraints) are assigned a higher quality. Relative importance between the constraints may also be specified.

8.2.5 Implementing the Optimization Methods

The next step is to include automatic optimization methods from the literature into the framework. This involves the following sub-steps:

1. Changing the optimization methods to support focus and user constraints. General approaches for this step were discussed in Section 8.1.

2. Changing the methods to consider information about an existing solution when computing a new solution. The emphasis here is to allow a cyclic improvement of the working solution,

as mentioned in Section 3.1.5.

3. If a method runs for a long period of time, then implementing a way of querying it for intermediate solutions during the processing. The method should also provide indication about its progress, that is, the amount of time which the user has to wait for the end of the computation; this information may be based on an analysis of the stop condition of the method.

4. Integrating the methods with the framework. This includes: creating graphical objects on the user interface and modules for choosing, running and stopping the methods; defining whether the methods run in parallel or alternately with user actions; defining whether and in which frequency intermediate solutions are sampled; specifying the conditions for replacing the working solution with intermediate and final solutions produced by the methods; and providing a visualization of the progress of the computation.

The data structure to be manipulated by the methods and other elements of the framework during the optimization process must also be designed at this stage.

8.2.6 Evaluating the Framework

Finally, the framework must be evaluated to verify whether human-computer collaboration in the new interactive system provides better results than the traditional way of solving the optimization problem. Such an evaluation may involve a comparison with a fully manual approach, with a fully automatic approach, or with manual post-processing optimization. Since the User Hints framework naturally supports all these modes of work, it can be used as a platform for testing all cases.

A sequence of steps for performing the evaluation is presented below:

1. Defining *major performance measures*. This step depends on the goal of the framework. If the goal is to help convergence, then typical performance measures are the quality of the final solution produced with the optimization process and/or the time needed to generate it. For a problem refinement task, the performance measure is related to the effectiveness in inserting domain knowledge into the optimization process.

2. Listing *basic measures* related to the use of hints and of other elements of the framework that can be evaluated. A number of basic measures are presented in the next section.

3. Designing test scenarios for comparing human-computer collaborative optimization against other approaches. The common scenarios are: (a) allowing only manual changes of the

solution, (b) executing the optimization methods without user interaction, and (c) allowing the user to perform manual changes and to use any automatic tools. Note that scenario (c) corresponds to the collaborative optimization, which we may hope to provide better results than the other options. These scenarios can be subdivided into simpler configurations, in order to test the basic measures individually, or to test combinations of them. However, testing all possibilities may be impossible, or even unnecessary. For example, a scenario involving only manual change is not useful for complex problems, for which no good solution can be produced without the help of an automatic method.

4. Deciding the type of evaluation. Two main alternatives are:

   - A quantitative evaluation, involving human experiments. This is the most common approach when the goal of the framework is to help convergence. Examples of this type of evaluation are presented by the Graph Drawing case studies, in Chapters 5 and 6.
   - Informative evaluation with domain experts, such as the heuristic evaluation method. This was done for the Map Labeling case study, in Chapter 7.

5. Preparing and performing the evaluation. This step usually involves a number of subtasks such as choosing the problem instances to be used in the evaluation, experimental design from the statistical point of view, recruiting users, providing training on the framework, and executing and recording the evaluation. Such tasks are usually supported by scientific background in human experiments and usability studies [144, 142, 171].

6. Analyzing the results. As we mentioned before, we hope to obtain better solutions by having collaborative optimization than by either a fully manual or a fully automatic approach. If the results are in agreement with this expectation, then the framework achieves its goal. On the other hand, negative results for a particular basic measure or for the entire experiment lead to two possible interpretations:

   - The user hints or the visualizations are not effective – then they should be redesigned or discarded. An example is the visualization of the conflict graph in the Map Labeling problem in Chapter 7, which was not helpful. When many negative results are observed, we can also question our conclusion from Section 8.2.1, that the problem was suitable for interactive optimization.

- User hints are useful, but interaction was not exploited correctly – this may suggest a better approach for combining human and computer collaboration. This was the case, for example, for the evaluation of the Graph Drawing framework in Chapter 6. We verified that human interaction was more effective for helping the genetic algorithm to escape from local minima. Therefore, new systems should start by using the genetic algorithm at the beginning of the optimization process as much as possible.

Even though the evaluation is listed here as the last step, it may be carried out in an iterative process, where visualizations and interactive functions of the framework are incrementally proposed and immediately tested.

Basic Measures

When performing the evaluation we should verify the effectiveness and efficiency of the elements of the framework such as the visualizations and the types of user hints. Some basic measures for these elements are listed below:

- *Constraint effectiveness* – indicates whether domain knowledge can be properly represented by user constraints, and whether the optimization methods are capable of dealing with these constraints.

- *Focusing effectiveness* – indicates whether focusing the optimization methods helps to explore the space of solutions. Scott *et al.* [170] investigate this issue in the HuGS paradigm by comparing a focused approach with a fully automatic algorithm. This is similar to our experiments in Chapter 6. Scott also investigated whether users could correctly identify areas of the solution for focus; they computed, among other factors, the percentage of focused areas that correctly yielded improvements, and the percentage of areas that could be improved but were not focused. Another way of evaluating focus is to compare the results produced by user-defined focus with the results of an automatic (possibly random) focus approach. In the latter case, the system automatically selects variables and runs an optimization method on them. We note that focus effectiveness depends on whether the user selects the minimum number of variables of the problem that produces the highest improvement of the solution. Selecting too few variables may reduce the flexibility of reorganizing the solution; on the other hand, selecting too many variables creates a larger space of solutions than necessary, which may slow down the search. The measure of effectiveness of focus, therefore, should ideally test whether the user achieves a good compromise between processing time and solution quality.

- *Manual-change effectiveness* – verifying whether manual changes directly improves constraint satisfaction or provides better results according to the objectives of the problem; it should consider whether manual changes help the optimization methods to escape from local minima, and to reduce processing time in general.
- *Appropriate stop timing* – identifying whether the user can stop the optimization method before a search for an optimal solution stagnates. This issue was investigated by Anderson *et al.* [8] for the HuGS paradigm, by investigating what improvement could be obtained if the algorithms were left running for a longer period of time. We did not do a formal evaluation of this aspect in this thesis. Nevertheless, we did notice that the users tended to stop the optimization method after a period of time without significant improvement of the solution (between 12 to 24 seconds, with our Genetic Algorithm for drawing graphs).
- *Effective visualization* – identifying whether the visualizations and the interaction operations defined on them are helpful for the user. This aspect may involve checking whether the most important elements and qualitative aspects of the optimization process were modeled by graphical objects. In addition, we should investigate whether interactive operations are supported by the visualization. More general measures, related to principles for designing meaningful, intuitive, and legible visualizations can be abstracted from the area of Information Visualization [31, 141].
- *Progressive learning* – investigating whether the effectiveness of the user actions increases with time. The users may learn effective ways of applying the optimization methods and performing manual changes. They can also become familiar with the visualization, and learn how to better interpret it. Learning was observed in the evaluations of our study cases through interviews. Scott *et al.* [170] discuss this issue as well, for the HuGS paradigm.
- *General satisfaction* – a final aspect is to investigate the user satisfaction with the framework. This not only indicates whether the user feels that the task was successfully completed, but also measures usability aspects of the interface.

Some of the basic measures are simple to define formally and evaluate in a quantitative way. Other basic measures, however, depend on a number of factors that cannot be controlled or predicted, or are subjective to the user. For the latter cases, the evaluation can be executed by having the users giving an informal opinion about the measures.

8.3 Interactive Versus Automatic Optimization

For the goal of improving convergence alone, we can question the value of implementing user interaction. A basic discussion point is: why invest effort in an interactive framework, if the optimization problem is well known and properly formulated? Why not develop better algorithms, that can solve the problem effectively and in a suitable amount of time, instead of asking a user to do the work? One may say that if humans are necessary for the optimization task, then it is because the existing algorithms are not good enough, and they should be improved. When considering costs, this argument can be much stronger. We may realize, for example, that developing a new algorithm (that can automate a process) involves a large investment at the beginning, but in the long term it provides a better tradeoff than expending money with workers' salaries for several years. Moreover, humans get sick, have all sorts of emotional problems, need breaks and take holidays, have variable performance, and may be simply not suitable for the job.

While all these arguments appear valid, they do not consider a few important engineering and political aspects:

- Developing a better algorithm can be quite difficult and very time consuming; some companies may not be willing to wait months or even years for a new technological solution. Rather, they may prefer to have humans filling the gap left by the existing optimization methods, until one day a fully automatic approach is available. Moreover, it may be the case that there is no guarantee that a computational solution will ever exist, due to the complexity of the problem and the limitations of computers, based on the Turing Machine model. Thus, humans, who are possibly not bounded by the same limitations, may offer more powerful resources to solve some problems.

- Even when very effective algorithms exist, the optimization problem itself may demand powerful computers, with fast CPUs and large memory capacity, in order to be solved. Such hardware technology may be too expensive for ordinary companies for the moment[1].

[1]Also note that, although it is against the current scientific and technological trends, some governments may prefer to promote processes and services that employ people, rather than ones that release them from their jobs. This attitude may be taken, for example, to reduce unemployment rates or to avoid intensive importation of foreign technology.

These aspects show the difficulty in adopting a fully automatic solution. However, we believe that investing in interactive optimization and in fully automatic optimization are not mutually exclusive choices; on contrary, they can complement each other (except in cases where there are limited budget and time constraints). On one hand, human interaction contributes to the performance and to the effectiveness of existing optimization methods, as we shown in this thesis. On the other hand, new and more advanced methods added to the optimization process can leverage human performance. The first advantage of having a more advanced optimization method is that it can produce initial solutions of much higher quality. Besides, a new method may also serve as a more effective tool for working on existing solutions. This effect was noticed, for example, during our experiments with the Graph Drawing problem. We realized that a genetic algorithm was not only capable of providing much better solutions automatically; it could also improve user-generated solutions in a more effective way than the heuristics in the Sugiyama method presented in Chapter 5.

The benefit of having both interactive facilities and very good automatic tools suggests extending the optimization module of the User Hints framework continuously. Initially, only few optimization methods are available, and the user may have to tackle several problems that cannot be solved automatically. Later, as more advanced optimization methods are developed, they are used to provide a better initial solution, and to help the user in a collaborative work mode.

8.4 Learning from Observation

Scott *et al.* [170] suggest that their systems could be used to train novice users on problem solving based on the HuGS paradigm. We recognize the same potential in the User Hints framework. Nonetheless, we can identify another interesting application, which characterizes the opposite learning process: instead of using the framework for training users, we can exploit it as an environment for helping us (developers of technology) to learn new algorithms. This can be done by observing expert users working on optimization problems, and then trying to abstract new algorithms based on the users' expertise and subjective strategies. The knowledge obtained with this process can be used later to automate activities in the framework that are currently executed by humans.

A simple approach for identifying new algorithms is to study manual changes of the working solution performed by the users. Such changes may represent promising strategies for solving problems that cannot be treated effectively by the optimization methods. For example, in the evaluation sections of Chapters 6 and 7 we identified some user heuristics. Those heuristics could be useful for implementing new automatic tools, which can be activated by the users for solving particular problems. It may also be possible to implement the heuristics as subroutines of a meta-heuristic method (for example, as a new mutation operator in a genetic algorithm) in order to improve its convergence.

A more automatic approach would be to investigate machine learning techniques for replacing the user. The aim here is to learn the user's strategies for employing semi-automatic tools of the framework, such as performing focus, and choosing, executing and stopping an optimization method. Research on learning human actions has been done for other domains [76, 194], and may provide useful ideas for exploring this issue in the User Hints framework.

We can also design agents in a *mixed initiative* [90] fashion that continuously observe the user and construct a model of his or her intentions. This model may help the user in performing frequent and repetitive tasks. For example, if an action a is almost always succeeded by an action b, then the system may propose the execution of b (or automatically execute it) when the user performs a.

Implementing some of the approaches and ideas mentioned above is quite challenging, and would constitute a new thesis by itself.

8.5 Extensions to the User Hints Framework

The User Hints framework can be extended in many ways.

An intuitive extension is to have more than one instance of the elements of the framework. Note that we have already been using several visualizations and optimization methods. We may support, in addition, more than one working solution simultaneously. These solutions could be created by different optimization methods, or by the same method using different controlling parameters (for example, two genetic algorithms with distinct population sizes or stop criteria). The user could then improve the solutions manually or reapply the optimization methods to do them. For a multi-objective problem, this extension may provide an interesting way for producing and displaying compromise solutions. Furthermore, interactive tools could be created to generate new solutions by merging parts of existing results.

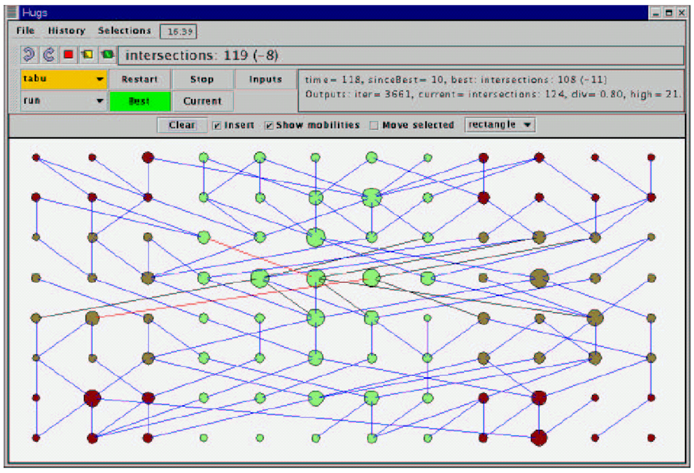
Another extension is to implement a history of previous working solutions, so that the user can revert to an "old" result (as in the HuGS approach) when the current optimization process does not lead to a promising end. The history does not have to save all working solutions generated during the optimization process, because this could demand much memory. Moreover, many solutions are not useful for the users, such as bad intermediate results generated during an improvement task. A possibility is, therefore, to construct a *selective* history. For example, we could save only the best solutions updated during the optimization process; in our human experiments in Chapter 5, the best solutions represented on average 6% of the total number of generated solutions.

Note that the best solution agent can be redesigned to be more informative. An extended agent may not only provide feedback about the improvement of the best solution, but also identify where and what type of improvement was obtained. For the GDHints system, for example, the agent could say “Good. Two edge crossings were eliminated”, and indicate this by showing a faded background image of the previous working solution. If it is desirable, then an avatar may be implemented to make the agent more anthropomorphic. A less invasive option, however, would be highlighting in a different color the information related to the improvement of edge crossings in the status bar.

The idea of keeping previous solutions suggests a new visualization for describing the direction of the optimization process. This visualization could display attributes of all working solutions produced over the time. A typical example is an *improvement-time chart*, as the one shown in Figure 5.10 on page 87, which describes improvements of the working solution for the edge crossing criterion. Such a visualization does not demand much memory, since it is sufficient to save only the quality parameters of the solutions.

An *improvement-time chart* can be generated even if the quality attribute is not a single value, but a vector of measures such as the entire quality vector in our GDHints system. In that case, we rank the quality vectors of all solutions during the optimization, and then create an X, Y-line chart showing the position of the solutions in the rank. Each solution has a point in the chart, whose Y-coordinate is given by its rank, and the X-coordinate is the time when the solution was produced. Note that the Y-coordinate of this visualization displays a relative distance between the solutions, rather than an absolute difference of their quality measures. Moreover, as new solutions are created during the optimization process, the rank and the visualization need to be updated. We believe that similar types of visualization, that display multiple solutions, may provide useful feedback to the user.

Another possible extension to the User Hints framework is to implement the *levels of mobility* used in the HuGS paradigm. This implementation may be different for each optimization problem, but in some cases mobility can be partially modeled by a combination of focus and constraint operations. An example is the Edge Crossing Minimization problem described by Klau *et al.* [110], where an improvement algorithm swaps two vertices in the same layer if at least one of the vertices has high mobility and none of them has low mobility (see Figure 8.2). Using our GDHints system based on the User Hints framework, high and medium mobility vertices would be set as selected, while low mobility vertices would stay unselected. The Left-Right constraints would then be automatically created for connecting low and medium mobility vertices, imposing an ordering between them. Optimization methods applied to this configuration would tend to reorganize the "high mobility" vertices, and preserve the sequential ordering of the remaining elements. The modeling, however, is not complete because the ordering of the constraints could be temporarily violated (except if we changed the optimization method to disallow any move that violates a Left-Right constraint). In addition, the algorithms could still spend time trying to move the "medium mobility" vertices, instead of concentrating action only on the "high mobility" ones, since no internal difference would exist between them. Thus, a proper implementation of levels of mobility would require addressing these issues.



```
intersections: 119 (-8)
time= 118, sinceBest= 10, best: intersections: 108 (-11)
Outputs: iter= 3661, current= intersections: 124, div= 0.80, high= 21.
```

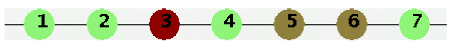


Figure 8.2: Levels of mobility for the Edge Crossing Minimization problem, from [110]. Low mobility vertices are shown in red (vertex 3 above), medium mobility vertices are in yellow (vertices 5 and 6), and high mobility vertices are in green. Courtesy of Joe Marks.

CHAPTER 9

Conclusion

This thesis demonstrates how human interaction can be useful in optimization processes. We investigate two major interactive goals: *refining the optimization problem* and *helping convergence to optimal solutions*.

An interactive framework is developed that allows a user to control the optimization processes. User actions are called *hints*, and include operations such as constraint adjustment, focus of optimization methods and manual changes of an existing solution.

Experiments with systems based on the User Hints framework demonstrate that human-computer collaboration can help to obtain better solutions than a fully automatic approach or a manual post-processing approach.

The effectiveness of the user interaction varies for the major interactive goals.

For the goal of *refining the problem*, we have found that:

- Human intervention is certainly necessary, and adding or removing constraints seems to be a promising way to insert domain knowledge. However, we did encounter some difficulties in using constraints. One of the challenges is to ensure that the set of constraints implemented in the system covers most of the user needs for changing the optimization problem dynamically. This is difficult, because adjustments of the problem may require new variables, conditions, and preferences that were unknown until runtime and cannot be entirely predicted. It is also necessary to guarantee that the user's interpretation of what a constraint does really matches its effect on the system. We noticed, for example, that the users in the experiments in Chapter 5 could not obtain the precise result they expected by using Left-Right constraints. In addition, modifying existing optimization methods to treat constraints effectively and efficiently can be a difficult task.

- The user can always change the solution manually to make it consistent with some domain knowledge. Nevertheless, this does not represent a permanent solution. In fact, changes performed by the user can be lost by running the optimization methods. The best way of using manual changes for problem refinement is either performing them in a post-processing stage only, or combining these changes with other types of user hints such as focus and constraint adjustment. In our Map Labeling system, for example, moving a label by hand to a new position causes a customized candidate position to be created or updated to reflect the placement. Also, the cost of the customized label position is set to zero, so that it has a high preference when using an optimization method. The main point here is to avoid situations where the optimization methods overrule the changes made by the user.

Regarding the goal of *helping convergence*, the investigation showed that:

- Manual change is important to improve the final quality of solutions. This was verified in particular when optimization methods became stuck in local minimal states. For other situations, mostly involving large problem instances (such as the problem of drawing the largest graphs in Chapter 6), a solution of higher quality can be obtained by running an optimization method alone rather than by having human intervention.
- Constraints by themselves are not effective in helping convergence. The benefits of this type of hint were very small and were realized only when combined with focus.
- Focus demonstrates a great potential for concentrating the action of the optimization methods, and thus, reducing the space of solutions to be explored. Nonetheless, the decision to focus on a small set of variables or on a large set is critical, since it may affect the final quality of the solution. Another interesting observation is that the basic feature for implementing focus – selection of variables – is an intuitive tool for concentrating any procedure (not only optimization methods) on a particular region of the problem. This was shown in Chapter 7 for changing the candidate positions of a set of selected point features.

Our experiments with the interactive systems suggest that the best way of exploiting the User Hints framework is first to execute the optimization methods to create an initial solution of high quality. The user can then perform manual changes and reapply the methods for further improvements. This is not, however, a strict recommendation as the framework supports several work modes and can be used in many different ways.

Finally, a philosophical comment about the future of interactive optimization is appropriate. As more effective and efficient optimization methods are developed, as well as artificial intelligent techniques combined with user profiling approaches, many activities executed by humans tend to be automated. One question is whether there will always be a place for interactive optimization.

Some people say that humans will never be fully replaced by computers, because computers are restrained by their programming code and hardware limitations which are different from the human "mental architecture". Even though this may be true, we believe that a more enlightening indication cames from the field of Economics. Mabry and Sharplin, in the article "Does more technology create unemployment?" [128], argue that unemployment caused by new technology is not so harmful for the economy as many people say. In fact, they show that the introduction of technology increases production of goods; such increase by itself creates new jobs in the business and service sectors in the medium and long terms, which finally absorbs professionals discharged from the automated sectors. In order words, technology discharges employees, but also creates new needs and new jobs (the same idea is discussed in [1]). While this argument may be contentious in general, our belief is that such reasoning is valid for interactive optimization. Better automatic tools allow us to grasp new concepts and to envisage new applications that involve more complex problems, which we did not have or could not understand before. The new problems by themselves are too complex for the current technology and need human expertise and intuition. Thus, a cyclic process exists, where we create automatic tools that help us to investigate new problems for which the existing technology is always one step behind. Such a process may ensure the presence of users in interactive optimization processes.

9.1 Future Research

We intend to continue the investigation of the User Hints framework. Our future projects include:

- Applying the User Hints framework to other optimization problems, in particular to Bin-packing problems and to the problem of drawing very large graphs such as biochemical pathways [102, 136].
- Analyzing the advantages and disadvantages of the extensions to the framework proposed in Section 8.5.
- Investigating approaches for observing and automating the users' actions as suggested in

Section 8.4. Machine learning and mixed initiative techniques can be explored.

- Experimenting with different dynamics for human-computer collaboration where automatic tools play a more active role, without reducing user interaction. It may be possible, for example, to have a system detecting areas of the working solution that need improvement, and automatically running an optimization method on them. Synchronization and locking mechanisms may be necessary to maintain consistency of the solution while a user and the system are working in parallel.

- Developing new general visualizations that can increase the understanding of complex optimization processes, and/or provide a higher degree of flexibility in directly manipulating the variables of the problem.

- Studying ways of applying and possibly adjusting the framework to support collaborative optimization with two or more users.

Bibliography

[1] “Technology and unemployment: A world without jobs?” *The Economist*, Vol. 334, No. 7901, February 1995, pp. 21–23.

[2] Aarts, E. and Korst, J., *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons Ltd., 1989.

[3] Abu-Mostafa, Y. S., “Machines that learn from hints,” *Scientific American*, April 1995, pp. 68–73.

[4] Ahn, J. and Freeman, H., “AUTONAP—an expert system for automatic map name placement,” *Proc. Int. Symp. Spatial Data Handling (SDH’84)*, 1984, pp. 544–569.

[5] Alpert, C. and Kahng, A., “Recent directions in netlist partitioning: A survey,” *Integration: The VLSI Journal*, Vol. 19, 1995.

[6] Alpert, C. J., “The ISPD circuit benchmark suite,” *Proceedings of the International Symposium on Physical Design (ISPD-98)*, ACM Press, New York, Apr. 6–8 1998, pp. 80–85.

[7] Andalman, B., Ryall, K., Ruml, W., Marks, J., and Shieber, S., “Design gallery browsers based on 2D and 3D graph drawing,” *Graph Drawing (Proc. GD ’97)*, edited by G. Di Battista, Vol. 1353 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 1997, pp. 322–329.

[8] Anderson, D., Anderson, E., Lesh, N., Marks, J., Mirtich, B., Ratajczak, D., and Ryall, K., “Human-guided simple search,” *AAAI/IAAI*, 2000, pp. 209–216.

[9] Arnold, M. H. and Scott, W. S., “An interactive maze router with hints,” *Proceedings of the 25th ACM/IEEE Design Automation Conference*, edited by ACM/IEEE, IEEE Computer Society Press, Anaheim, CA, Jun. 1988, pp. 672–676.

[10] Bachmann, T., *Ein interaktives Planungssystem zur Dienst – und Umlaufplanung für die Stadtwerke Passau*, Master’s thesis, Universität Passau, 1992, Diplomarbeit.

[11] Baker, B. S., “Approximation algorithms for NP-complete problems on planar graphs,” *J. ACM*, Vol. 41, 1994, pp. 153–180.

[12] Barbosa, H. J. C. and Barreto, A. M. S., “An interactive genetic algorithm with co-evolution of weights for multiobjective problems,” *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, edited by L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Morgan Kaufmann, San Francisco, California, USA, 7-11 Jul. 2001, pp. 203–210.

[13] Bayazit, O. B., Song, G., and Amato, N., “Providing haptic ‘hints’ to automatic motion planners,” *Phantom Users Group Workshop (PUG99)*, MIT, Cambridge, MA, October 1999.

[14] Bayazit, O. B., Song, G., and Amato, N. M., “Enhancing randomized motion planners: Exploring with haptic hints,” *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA’00)*, April 2000, pp. 529–536.

[15] Beasley, J. E., “Algorithms for unconstrained two-dimensional guillotine cutting,” *Journal of Operational Research Society Ltd*, Vol. 36, No. 4, 1985, pp. 297–306.

[16] Bellman, R., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[17] Bentley, P. J., “Exploring component-based representations – the secret of creativity by evolution?” *In Proc. of the Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2000)*, University of Plymouth, UK, April 2000.

[18] Berard, C. and Grislain, J., “GET: an interactive timetabling system using CIM concepts,” *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT ’95)*, 1995, pp. 523–533.

[19] Bern, M. and Eppstein, D., “Approximation algorithms for geometric problems,” *Approximation Algorithms for NP-Hard Problems*, edited by D. S. Hochbaum, PWS Publishing Company, Boston, MA, 1997, pp. 296–345.

[20] Biedl, T. and Brandenburg, F., “Graph drawing contest report,” *Graph Drawing (GD’01)*, Vol. 2265 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002, pp. 513–521.

[21] Bloem, R., Ravi, K., and Somenzi, F., “Symbolic guided search for CTL model checking,” *Design Automation Conference*, 2000, pp. 29–34.

[22] Böhringer, K.-F. and Paulisch, F. N., “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*, 1990, pp. 43–51.

[23] Bollobás, B., *Extremal Graph Theory*, Academic Press, New York, 1978.

[24] Bolz, D., “Some aspects of the user interface of a knowledge based beautifier for drawings,” *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, Session 1: Intelligent Presentation, 1993, pp. 45–52.

[25] Brandenburg, F. J., Professor, Faculty of the Department of Mathematics and Informatics. The University of Passau. Private communication, 2002.

[26] Brandenburg, F. J., “Graph drawing contest report,” *Graph Drawing (GD’02)*, Vol. 2528 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002, pp. 376–379.

[27] Brandenburg, F. J., Brandes, U., Himsolt, M., and marcus Raitner, “Graph drawing contest report,” *Graph Drawing (GD’00)*, Vol. 1984 of *Lecture Notes in Computer Science*, Springer-verlag, 2001.

[28] Brandenburg, F. J., Jünger, M., Marks, J., Mutzel, P., and Schreiber, F., “Graph drawing contest report,” *Graph Drawing (GD’99)*, Vol. 1731 of *Lecture Notes in Computer Science*, Springer-Verlag, 1999, pp. 400–409.

[29] Branke, J., Bucher, F., and Schmeck, H., “A genetic algorithm for drawing undirected graphs,” *Proc. 3rd Nordic Work. Genetic Algorithms and Their Applications, 3NWGA*, Finnish Artificial Intelligence Society, 20–22 Aug. 1997, pp. 193–206, Paper 15.

[30] Bui, T. N. and Moon, B. R., “Genetic algorithm and graph partitioning,” *IEEE Transactions on Computers*, Vol. 45, No. 7, 1996, pp. 841–855.

[31] Card, S. K., MacKinlay, J. D., Shneiderman, B., and Card, M., *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Series in Interactive Technologies, Academic Press, 1999.

[32] Chazelle, B. and 36 co-authors, “The computational geometry impact task force report,” *Advances in Discrete and Computational Geometry*, edited by B. Chazelle, J. E. Goodman, and R. Pollack, Vol. 223, American Mathematical Society, Providence, 1999, pp. 407–463.

[33] Christensen, J., Marks, J., and Shieber, S., “Placing text labels on maps and diagrams,” *Graphics Gems IV*, edited by P. Heckbert, Academic Press, Boston, MA, 1994, pp. 497–504.

[34] Christensen, J., Marks, J., and Shieber, S., “An empirical study of algorithms for point-feature label placement,” *ACM Trans. Graph.*, Vol. 14, 1995, pp. 202–232.

[35] Cook, S. A., “The complexity of theorem-proving procedures,” *Conference record of third annual ACM Symposium on Theory of Computing: papers presented at the symposium, Shaker Heights, Ohio, May 3, 4, 5, 1971*, edited by ACM, ACM Press, New York, NY, USA, 1971, pp. 151–158.

[36] Cook, W., “Solving traveling salesman problems,” *Lecture Notes in Computer Science*, Vol. 2461, 2002.

[37] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[38] Davidson, R. and Harel, D., “Drawing graphs nicely using simulated annealing,” Tech. rep., Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.

[39] Davidson, R. and Harel, D., “Drawing graphics nicely using simulated annealing,” *ACM Trans. Graph.*, Vol. 15, No. 4, 1996, pp. 301–331.

[40] de Mendonça N., C. F. X., *A layout system for information system diagrams*, Ph.D. thesis, Department of Computer Science, University of Queensland, 1994.

[41] de Souza, P. S. and Talukdar, S. N., “Genetic algorithms in asynchronous teams,” *Proc. of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, 1991.

[42] de Souza, P. S. and Talukdar, S. N., “Asynchronous organizations for multi-algorithm problems,” *Proc. ACM Symposium on Applied Computing*, Indianapolis, IN, February 1993.

[43] Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G., *Graph Drawing*, Prentice Hall, Upper Saddle River, NJ, 1999.

[44] Dietz, P. and Leigh, D., “Diamondtouch: A multi-user touch technology,” *Proceedings of UIST ’01*, ACM Press, 2001, pp. 219–226.

[45] Djouadi, Y., “Cartage: A cartographic layout system based on genetic algorithms,” *Proc. EGIS’94*, 1994, pp. 48–56.

[46] do Nascimento, H. A. D., *Uma Abordagem para Desenho de Grafos Baseada na Utilização de Times Assíncronos (“An Approach for Drawing Graphs using Asynchronous Teams”)*, Master’s thesis, Instituto de Computação - UNICAMP, Brazil, May 1997.

[47] do Nascimento, H. A. D., “A framework for human-computer interaction in directed graph drawing,” *Australian Symposium on Information Visualisation*, Vol. 9, Australian Computer Society Inc., 2001, pp. 63–69.

[48] do Nascimento, H. A. D. and Eades, P., “User Hints for directed graph drawing,” *Graph Drawing (Proc. GD ’01)*, Vol. 2265 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 2001, pp. 205–219.

[49] do Nascimento, H. A. D. and Eades, P., “A focus and constraint-based genetic algorithm for interactive directed graph drawing,” *Soft Computing Systems - Design, Management and Applications*, edited by A. Abraham, J. Ruiz-del-Solar, and M. Köppen, Frontiers in Artificial Intelligence and Applications Vol. 87, IOS Press Amsterdam, Berlin, Oxford, Tokyo, Washington D.C., 2002, pp. 634–643.

[50] do Nascimento, H. A. D. and Eades, P., “A focus and constraint-based genetic algorithm for interactive directed graph drawing,” Tech. Report TR533, School of Information Technologies, The University of Sydney, Australia, 2002.

[51] do Nascimento, H. A. D. and Eades, P., “User Hints for map labelling,” *Twenty-Sixth Australasian Computer Science Conference (ACSC2003)*, edited by M. J. Oudshoorn, Vol. 16 of *Conferences in Research and Practice in Information Technology*, ACS, Adelaide, Australia, 2003, p. 339.

[52] do Nascimento, H. A. D., Eades, P., and de Mendonça N., C. F. X., “A multiagent approach using A-Teams for graph drawing,” Technical Report 2000-02, Department of Computer Science and Software Engineering, University of Newcastle, Callaghan 2308, Australia, 2000.

[53] do Nascimento, H. A. D., Eades, P., and de Mendonça Neto, C. F. X., “A multiagent approach using A-Teams for graph drawing,” *Proceedings of the 9th International Conference on Intelligent Systems*, Louisville, Kentucky – USA, 2000, pp. 39–42.

[54] Doerschler, J. S., “Data structures required for overlap detection in an expert map name placement system,” Tech. Rep. IPL-TR-077, Image Processing Laboratory, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, N.Y. 12181, 1985.

[55] Doerschler, J. S. and Freeman, H., “An expert system for dense-map name placement,” *Proc. Auto-Carto 9*, 1989, pp. 215–224.

[56] Doğrusöz, U., Kakoulis, K. G., Madden, B., and Tollis, I. G., “Edge labeling in the graph layout toolkit,” *Proceedings of the Symposium on Graph Drawing (GD’98)*, Vol. 1547 of *Lecture Notes in Computer Science*, Springer-Verlag, Aug. 1998, pp. 356–363.

[57] Eades, P., “A heuristic for graph drawing,” *Congr. Numer.*, Vol. 42, 1984, pp. 149–160.

[58] Eades, P. and Lin, X., “How to draw a directed graph,” *Proc. IEEE Workshop on Visual Languages*, 1989, pp. 13–17.

[59] Eades, P. and Marks, J., “Graph drawing contest report,” *Graph Drawing (GD’94)*, Vol. 894 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 143–146.

[60] Eades, P., Marks, J., and North, S., “Graph drawing contest report,” *Graph Drawing (GD’96)*, Vol. 1190 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pp. 129–138.

[61] Edmondson, S., Christensen, J., Marks, J., and Shieber, S., “A general cartographic labeling algorithm,” *Cartographica*, Vol. 33, No. 4, 1997, pp. 13–23.

[62] Eloranta, T. and Mäkinen, E., “TimGA - A genetic algorithm for drawing undirected graphs,” Tech. Rep. A-1996-10, Department of Computer Science, University of Tampere, 1996, A revised version will appear in Divulgaciones Matematicas.

[63] Elsner, U., “Graph partitioning: a survey,” Tech. Rep. Preprint SFB393/97-27, Technische Universität Chemnitz, Germany, Dec 1997.

[64] Faigle, U., Schrader, R., and Suletzki, R., “A cutting plane algorithm for optimal graph partitioning,” *Methods of Operations Research*, Vol. 57, 1987, pp. 109–116.

[65] Feo, T. A. and Resende, M. G. C., “A probabilistic heuristic for a computationally difficult set covering problem,” *Operations Research Letters*, Vol. 8, 1989, pp. 67–71.

[66] Feo, T. A. and Resende, M. G. C., “Greedy randomized adaptive search procedures,” *Journal of Global Optimization*, Vol. 6, 1995, pp. 109–133.

[67] Ferguson, G. M., Allen, J. F., Miller, B. W., and Ringger, E. K., “The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant,” Tech. Rep. TN96-5, 1996.

[68] Fiduccia, C. M. and Mattheyses, R. M., “A linear-time heuristic for improving network partitions,” *Proceedings of* $19th$ *Design Automation Conference*, ACM/IEEE, Las Vegas, Jun. 1982, pp. 175–181.

[69] Fjällström, P.-O., “Algorithms for graph partitioning: A survey,” *Linköping Electronic Articles in Computer and Information Science*, 1998.

[70] Formann, M. and Wagner, F., “A packing problem with applications to lettering of maps,” *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 1991, pp. 281–288.

[71] Foulds, L. R., *Combinatorial Optimization for Undergraduates*, Springer Verlag, New York, 1984.

[72] Freeman, H., “An expert system for the automatic placement of names on a geographic map,” *Information Sciences*, Vol. 45, 1988, pp. 367–378.

[73] Garey, M., Johnson, D., and Stockmeyer, L., “Some simplified NP-complete graph problems,” *Theoretical Computer Science*, Vol. 1, 1976, pp. 237–267.

[74] Garey, M. R. and Johnson, D. S., *Computers and Intractability*, W.H. Freeman and Co., New York, 1979.

[75] Garey, M. R. and Johnson, D. S., “Crossing number is NP-complete,” *SIAM J. Algebraic Discrete Methods*, Vol. 4, No. 3, 1983, pp. 312–316.

[76] Garland, A. and Lesh, N., “Learning Hierarchical Task Models by Demonstration,” Tech. Rep. TR2001-03, Mitsubishi Electric Research Laboratories, 2001.

[77] Geoffrion, A. M., “Lagrangian relaxation for integer programming,” *Mathematical Programming Study*, Vol. 2, 1974, pp. 82–114.

[78] Glover, F., “Tabu search - Part 2,” *ORSA Journal on Computing*, Vol. 2, No. 1, 1989, pp. 4–32.

[79] Glover, F., “Tabu search: 1,” *ORSA Journal on Computing*, Vol. 1, No. 3, Summer 1989, pp. 190–206.

[80] Godfrey, D., Director, Sydway Publishing Pty. Ltd., Australia. Private communication, 2002.

[81] Goldberg, D. E., *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison Wesley Publishing Company, 1989.

[82] Goltz, H.-J., “Combined automatic and interactive timetabling using constraint logic programming,” *Proceedings Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT 2000)*, edited by E. Burke and W. Erben, Konstanz, 2000, pp. 78–95.

[83] Goltz, H.-J. and Matke, D., “Combined interactive and automatic timetabling,” *In Proc. PACLP’99*, London, April 1999, pp. 529–535.

[84] Gomory, R. E., “Outline of an algorithm for integer solutions to linear programs,” *Bulletin of the American Mathematical Society*, Vol. 64, 1958, pp. 275–278.

[85] Gradišar, M., Jesenko, J., and Resinovic, G., “Optimization of roll cutting in clothing industry,” *Computers Operations Research*, Vol. 24, No. 10, 1997, pp. 945–953.

[86] Grötschel, M. and Lovász, L., *Combinatorial optimization*, Vol. 2 of *Handbook of Combinatorics*, chap. 28, North Holland, 1995, pp. 1541–1597.

[87] Grötschel, M. and Wakabayashi, Y., “Facets of the clique partitioning polytope,” *Mathematical Programming*, Vol. 47, 1990, pp. 367–387.

[88] Groves, L. J., Michalewicz, Z., Elia, P. V., and Janikow, C. Z., “Genetic algorithms for drawing directed graphs,” *Methodologies for Intelligent Systems, 5. Proceedings of the Fifth International Symposium*, edited by Z. W. Ras, M. Zemankova, and M. L. Emrich, North-Holland, Amsterdam, Netherlands, Knoxville, TN, 25-27 Oct. 1990, pp. 268–276.

[89] Gschwind, D. J. and Murtagh, T. P., “A recursive algorithm for drawing hierarchical directed graphs,” Tech. Rep. CS-89-02, Department of Computer Science, Williams College, 1989.

[90] Hearst, M. A., "Trends & controversies: Mixed-initiative interaction," *IEEE Intelligent Systems*, Vol. 14, No. 5, 1999, pp. 14–23.

[91] Hirsch, S. A., "An algorithm for automatic name placement around point data," *The American Cartographer*, Vol. 9, No. 1, 1982, pp. 5–17.

[92] Homaifar, Guan, and Liepins, "A new approach to the traveling salesman problem using genetic algorithms," *COMPSYSTS: Complex Systems*, Vol. 6, 1992.

[93] Hopper, E. and Turton, B., "A genetic algorithm for a 2D industrial packing problem," *Computers & Industrial Engineering*, Vol. 37, No. 1-2, 1999, pp. 375–378.

[94] Horvitz, E., "Principles of mixed-initiative user interfaces," *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, Vol. 1 of *Characters and Agents*, 1999, pp. 159–166.

[95] Imai, H. and Asano, T., "Efficient algorithms for geometric graph search problems," *SIAM J. Comput.*, Vol. 15, No. 2, 1986, pp. 478–494.

[96] Imhof, E., "Die Anordnung der Namen in der Karte," *International Yearbook of Cartography*, Vol. 2, Kirschbaum, Bonn Bad Godesberg, 1962, pp. 93–129.

[97] Imhof, E., "Positioning names on maps," *Amer. Cartogr.*, Vol. 2, 1975, pp. 128–144.

[98] Ingwersen, P., *Information Retrieval Interaction*, Taylor Graham, London, 1992.

[99] Jacobsen, A. E., *Interaktion und Lernverfahren beim Zeichnen von Graphen mit Hilfe evolutionärer Algorithmen (Interaction and learning methods for graph layouts with the help of evolutionary algorithms)*, Master's thesis, Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, 2001, Diplomarbeit.

[100] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C., "Optimization by simulated annealing: an experimental evaluation; part 1, graph partitioning," *Operations Research*, Vol. 37, No. 6, Nov. 1989, pp. 865–892.

[101] Jones, C., "Cartographic name placement with Prolog," *IEEE Comput. Graph. Appl.*, Vol. 5, 1989, pp. 36–47.

[102] Kanne, C.-C., Schreiber, F., and Trümbach, D., “Electronic biochemical pathways,” *Lecture Notes in Computer Science*, Vol. 1731, 2000, pp. 418–419.

[103] Karp, R. M., “Reducibility among combinatorial problems,” *Complexity of Computer Computations*, edited by R. E. Miller and J. W. Thatcher, Plenum Press, New York, 1972, pp. 85–103.

[104] Katchabaw, M. J., Lutfiyya, H. L., and Bauer, M. A., “Using user hints to guide resource management for quality of service,” *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, edited by W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Las Vegas, Nevada, June 1999, pp. 1069–1075.

[105] Kernighan, B. W. and Lin, S., “An efficient heuristic procedure for partitioning graphs,” *Bell System Technical Journal*, Vol. 49, No. 2, Feb. 1970, pp. 291–307.

[106] Kilander, F., Fåhraeus, E., and Palme, J., “Intelligent information filtering – the intfilter project,” Tech. Rep. 97-002, Depart. of Computer and Systems Sciences, Stockholm University, 1997.

[107] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., “Optimization by simulated annealing,” *Science*, Vol. 220, 1983, pp. 671–680.

[108] Klau, G. W., *A Combinatorial Approach to Orthogonal Placement Problems*, Ph.D. thesis, Fachbereich Informatik, Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, Sep. 2001.

[109] Klau, G. W., Lesh, N., Marks, J., and Mitzenmacher, M., “Human-guided tabu search,” *To appear in Proc. of AAAI 2002 (The Eighteenth National Conference on Artificial Intelligence*, 2002.

[110] Klau, G. W., Lesh, N., Marks, J., Mitzenmacher, M., and Schafer, G. T., “The hugs platform: A toolkit for interactive optimization,” *Proc. of the International Working Conference on Advanced Visual Interfaces (AVI 2002)*, edited by L. Terveen, D. Wixon, E. Comstock, and A. Sasse, Trento, Italy, 2002.

[111] Knuth, D. E., *The Art of Computer Programming, Volume 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, USA, 3rd ed., 1997.

[112] Knuth, D. E., *The Art of Computer Programming, Vol 3, Sorting and Searching*, Addison-Wesley, Reading, MA, USA, 2nd ed., 1998.

[113] Knuth, D. E., *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Addison-Wesley, Reading, MA, USA, 3rd ed., 1998.

[114] Kochhar, S., Marks, J., and Friedell, M., "Interaction paradigms for human-computer cooperation in graphical-object modeling," *Proceedings of Graphics Interface '91*, Jun. 1991, pp. 180–191.

[115] Költze, S., *Ein interaktives Planungssystem zur Gestaltung und Optimierung von Zeitplaenen am Beispiel von Wettkampfplänen in der Leichtathletik*, Master's thesis, Universität Passau, 1995, Diplomarbeit.

[116] Kosak, C. and Marks, J., "A parallel genetic algorithm for network-diagram layout," *Proc. 4th Internat. Conf. on Genetic Algorithms*, 1991.

[117] Koschke, R. and Eisenbarth, T., "A framework for experimental evaluation of clustering techniques," *Proc. of 8th International Workshop on Program Comprehension*, IEEE, June 2000, pp. 201–210.

[118] Koutsofios, E. and North, S., "Drawing graphs with *dot*," Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ., 1995, Available from `http://www.research.bell-labs.com/dist/drawdag`.

[119] Kulju, W. and Lutfiyya, H., "Design and implementation of an application layer protocol for reducing UDP traffic based on user hints and policies," *Lecture Notes in Computer Science*, Vol. 2496, 2002, pp. 263–275.

[120] Kumar, V., Grama, A., Gupta, A., and Karypis, G., *Introduction to parallel computing. Design and analysis of algorithms*, The Benjamin/Cummings Publishing Company, 1994.

[121] Kureichick, V., Miagkikh, V., and Topchy, A., "Genetic algorithm for solution of the traveling salesman problem with new features against premature convergence," TSURE Journal of Engineering, No. 3, 1997.

[122] Lalanne, D., Melissargos, G., and Pu, P., "Solving complex problems with computational and interfacing tools," *Swiss workshop on collaborative systems*, May 1997.

[123] Lesh, N., Research Scientist, Mitsubishi Electric Research Laboratories, Boston. Private communication, 2000.

[124] Lesh, N., Marks, J., and Patrignani, M., "Interactive partitioning," *International Symposium of Graph Drawing (Lecture Notes in Computer Science)*, Vol. 1984, September 2000, pp. 31–36.

[125] Levin, L. A., "Universal sorting problems," *Problemy Peredaci Informacii*, Vol. 9, 1973, pp. 115–116, In Russian. English translation in *Problems of Information Transmission* 9:265–266.

[126] Lo, L. K., "A genetic algorithm to solve the 2-D bin packing problem," *Genetic Algorithms and Genetic Programming at Stanford 1999*, edited by J. R. Koza, Stanford Bookstore, Stanford, California, 94305-3079 USA, 15 Mar. 1999, pp. 122–130.

[127] Louis, S. J. and Tang, R., "Interactive genetic algorithms for the traveling salesman problem," *Proceedings of the Genetic and Evolutionary Computation Conference*, edited by W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Vol. 1, Morgan Kaufmann, Orlando, Florida, USA, 13-17 1999, pp. 385–392.

[128] Mabry, R. H. and Sharplin, A. D., "Does more technology create unemployment?" Cato Policy Analysis, No. 68, March 1986.

[129] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., and Shieber, S., "Design galleries: A general approach to setting parameters for computer graphics and animation," *SIGGRAPH 97 Conference Proceedings*, edited by T. Whitted, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, Aug. 1997, pp. 389–400, ISBN 0-89791-896-7.

[130] Marks, J. and Shieber, S., "The computational complexity of cartographic label placement," Tech. Rep. TR-05-91, Harvard CS, 1991.

[131] Massie, T. H. and Salisburg, J. K., "The phantom haptic interface: A device for probing virtual objects," *Proceedings of the ASME International Mechanical Engineering Congress and Exhibition*, Vol. DSC 55-1, Chicago, IL, 1994, pp. 295–302.

[132] Masui, T., "Graphic object layout with interactive genetic algorithms," *Proc. of the 1992 IEEE Workshop on Visual Languages*, Seattle, WA, 1992, pp. 74–80.

[133] Masui, T., “Evolutionary learning of graph layout constraints from examples,” *Proc. of the 7th Annual Symposium on User Interface Software and Technology (UIST’94)*, Marina del Rey, CA, 1994, pp. 103–108.

[134] Merz, P. and Freisleben, B., “Genetic Local Search for the TSP: New Results,” *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, edited by T. Bäck, Z. Michalewicz, and X. Yao, IEEE Press, Piscataway, NJ, USA, 1997, pp. 159–164.

[135] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., “Equation of state calculations by fast computing machines,” *Journal Chem. Physics*, Vol. 6, No. 21, 1953, pp. 1087–1092.

[136] Michal, G., editor, *Biochemical Pathways*, Spektrum Akademischer Verlag, Heidelberg, 1999.

[137] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, 2nd ed., 1994.

[138] Michalewicz, Z. and Fogel, B. F., *How to solve it: modern heuristics*, Springer-Verlag, 2000.

[139] Müller, T. and Barták, R., “Interactive timetabling,” *Proc. of the Sixth Annual Workshop of the ERCIM Working Group on Constraints*, Prague, Czech Republic, June 2001.

[140] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*, John Wiley and Sons, New York, 1988.

[141] Nesbitt, K. V., *Multi-sensory Display of Abstract Data*, Ph.D. thesis, School of Information Technologies, University of Sydney, Australia, January 2003.

[142] Newman, W. M. and Lamming, M. G., *Interactive System Design*, Addison-Wesley, MA, 1995.

[143] Neyer, G., “Map labeling with application to graph drawing,” *Drawing Graphs: Methods and Models*, edited by D. Wagner and M. Kaufmann, Vol. 2025 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001, pp. 247–273.

[144] Nielsen, J., *Usability Engineering*, AP Professional, Cambridge, 1993.

[145] North, S. C., editor, *Graph Drawing (Proc. GD '96)*, Vol. 1190 of *Lecture Notes Comput. Sci.*. Springer-Verlag, 1997.

[146] Paris, G., "Cooperation between interactive actions and automatic drawing in a schematic editor," *Graph Drawing (Proc. GD '98)*, edited by S. Whitesides, Vol. 1547 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 1998, pp. 394–402.

[147] Pfefferkorn, C., Burr, D., Harrison, D., Heckman, B., Oresky, C., and Rothermel, J., "ACES: A cartographic expert system," *Proc. Auto-Carto 7*, 1985, pp. 399–407.

[148] Poon, S.-H., Shin, C.-S., Strijk, T., and Wolff, A., "Labeling points with weights," Tech. Rep. 7/2001, Institut für Mathematik und Informatik, Universität Greifswald, May 2001.

[149] Poon, S.-H., Shin, C.-S., Strijk, T., and Wolff, A., "Labeling points with weights," *Abstracts 17th European Workshop Comput. Geom.*, Freie Universität Berlin, 2001, pp. 97–100.

[150] Pu, P. and Lalanne, D., "Human and machine collaboration in creative design," *European Conference on Artificial Intelligence*, 1996, pp. 276–282.

[151] Pu, P. and Lalanne, D., "Interactive problem solving via algorithm visualization," *INFOVIS*, 2000, pp. 145–154.

[152] Pu, P. and Lalanne, D., "Design visual thinking tools for mixed initiative systems," *Proceedings of the 2002 International Conference on Intelligent User Interfaces (IUI-02)*, edited by Y. Gil and D. B. Leake, ACM Press, New York, Jan. 13–16 2002, pp. 119–126.

[153] Purchase, H., "Which aesthetic has the greatest effect on human undestanding?" *Graph Drawing (Proc. GD '97)*, Vol. 1353 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 1997, pp. 248–261.

[154] Qin, Z., Wolff, A., Xu, Y., and Zhu, B., "New algorithms for two-label point labeling," Tech. Rep. HKUST-TCSC-2000-06, Hongkong University of Science and Technology, Jun. 2000.

[155] Raidl, G., "An evolutionary approach to point-feature label placement," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, edited by W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Morgan Kaufmann, Jul. 1999, p. 807.

[156] Raidl, G. R., “The multiple container packing problem: A genetic algorithm with weighted codings,” *ACRVW: Applied Computing Review: A Publication of the ACM Special Interest Group on Applied Computing*, Vol. 7, 1999.

[157] Rayside, D., Reuss, S., Hedges, E., and Kontogiannis, K., “The effect of call graph construction algorithms for object-oriented programs on automatic clustering,” *Proc. of 8th International Workshop on Program Comprehension*, IEEE, June 2000, pp. 191–200.

[158] Reeves, C., “Hybrid genetic algorithms for bin-packing and related problems,” *Annals of Operations Research*, Vol. 63, 1996, pp. 371–396.

[159] Rodrigues, R. F. and de Souza, P. S., “Asynchronous teams: a multi-algorithm approach for solving combinatorial multi-objective optimization problems,” *Proceedings of the 5th Workshop of the DGOR-Working Group Multicriteria Optimization and Decision Theory*, Germany, May 1995.

[160] Rolland, E., Pirkul, H., and Glover, F., “Tabu search for graph partitioning,” *Operations Research*, Vol. 63, 1997, pp. 209–232.

[161] Rosete-Suarez, A., Ochoa-Rodriguez, A., and Sebag, M., “Automatic graph drawing and stochastic hill climbing,” *Proceedings of the Genetic and Evolutionary Computation Conference*, edited by W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Vol. 2, Morgan Kaufmann, Orlando, Florida, USA, 13-17 Jul. 1999, pp. 1699–1706.

[162] Rosete-Suarez, A., Ochoa-Rodriguez, A., and Sebag, M., “Evolving objective function for saving computational effort: a case study,” *Proceedings of 2nd International Symposium on Articial Intelligence, ISAS’99, CIMAF’99*, edited by R. S. A. Ochoa, M. R. Soto, Havana, March 1999, pp. 174–181.

[163] Rosete-Suarez, A., Sebag, M., and Ochoa-Rodriguez, A., “A study of evolutionary graph drawing,” Tech. Rep. Rapport de Recherche 1228, LRI, Universite Paris-Sud XI, 1999.

[164] Russel, G., UBD, Universal Press, Australia. Private communication, 2002.

[165] Ryall, K., Marks, J., and Shieber, S., “An interactive constraint-based system for drawing graphs,” *Graph Drawing (Proc. GD ’96)*, Proc. of the ACM Symposium on User interface Software and Technology (UIST’ 97), Banff, Alberta, October 1997, pp. 97–104.

[166] Ryall, K., Marks, J., and Shieber, S., "An interactive system for drawing graphs," *Graph Drawing (Proc. GD '96)*, edited by S. North, Vol. 1190 of *Lecture Notes Comput. Sci.*, Springer-Verlag, 1997, pp. 387–393.

[167] Saab, Y. and Rao, V., "Stochastic evolution: A fast effective heuristic for some genetic layout problems," *Proc. 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 26–31.

[168] Salkin, H. M. and Mathur, K., *Foundations of Integer Programming*, North-Holland, New York, 1989.

[169] Sanchis, L. A., "Multiple-way network partitioning," *IEEE Transactions on Computers*, Vol. 38, No. 1, Jan. 1989, pp. 62–81.

[170] Scott, S. D., Lesh, N., and Klau, G. W., "Investigating human-computer optimization," *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, edited by L. Terveen, D. Wixon, E. Comstock, and A. Sasse, ACM Press, New York, Apr. 20–25 2002, pp. 155–163.

[171] Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing, Reading, MA, 2nd ed., 1992.

[172] Shneiderman, B. and Kang, H., "Direct annotation: A drag-and-drop strategy for labeling photos," *Proc. IEEE Int. Conf. on Information Visualisation (IV'00)*, edited by E. Banissi, M. Bannatyne, C. Chen, F. Khosrowshahi, M. Sarfraz, and A. Ursyn, London, 19–21 Jul. 2000, pp. 88–95.

[173] Smith, S., Lassila, O., and Becker, M., "Configurable mixed-initiative systems for planning and scheduling," *Advanced Planning Technology*, edited by A. Tate, AAAI Press, May 1996.

[174] Spence, R., *Information Visualization*, ACM Press, 2000.

[175] Stolfi, J., do Nascimento, H. A. D., and de Mendonça, C. F. X., "Heuristics and pedigrees for drawing directed graphs," *Journal of Brazilian Computer Society*, Vol. 6, No. 1, July 1999, pp. 38–49.

[176] Strijk, T., Verweij, B., and Aardal, K., "Algorithms for maximum independent set applied to map labelling," Tech. Rep. UU-CS-2000-22, Department of Computer Science, Utrecht University, 2000.

[177] Sugiyama, K. and Misue, K., "Visualization of structural information: Automatic drawing of compound digraphs," *IEEE Trans. Softw. Eng.*, Vol. 21, No. 4, 1991, pp. 876–892.

[178] Sugiyama, K., Tagawa, S., and Toda, M., "Methods for visual understanding of hierarchical systems," *IEEE Trans. Syst. Man Cybern.*, Vol. SMC-11, No. 2, 1981, pp. 109–125.

[179] Swamy, M. N. S. and Thulasiraman, K., *Graphs, Networks, and Algorithms*, John Wiley & Sons, New York, 1981.

[180] Takagi, H., "Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation," *Proc. of the IEEE*, Vol. 89, No. 9, 2001, pp. 1275–1296.

[181] Talukdar, S. N. and de Souza, P. S., "Scale efficienty organizations," *IEEE Int. Conference on Systems, Man and Cybernetics*, October 1992.

[182] Tamassia, R., Di Battista, G., and Batini, C., "Automatic graph drawing and readability of diagrams," *IEEE Trans. Syst. Man Cybern.*, Vol. SMC-18, No. 1, 1988, pp. 61–79.

[183] Tea, H., "Genetic algorithms applied to the traveling salesman problem," *Genetic Algorithms and Genetic Programming at Stanford 2000*, edited by J. R. Koza, Stanford Bookstore, Stanford, California, 94305-3079 USA, Jun. 2000, pp. 397–406.

[184] Tichy., W. F., "Should computer scientists experiment more?" *IEEE Computer*, Vol. 31, No. 5, May 1998, pp. 32–40.

[185] Tollis, I., Sydway Publishing Pty. Ltd., Australia. Private communication, 2002.

[186] Tzerpos, V. and Holt, R. C., "On the stability of software clustering algorithms," *Proc. of 8th International Workshop on Program Comprehension*, IEEE, June 2000, pp. 211–218.

[187] Utech, J., Branke, J., Schmeck, H., and Eades, P., "An evolutionary algorithm for drawing directed graphs," *In Proceedings of the 1998 International Conference on Imaging Science, Systems, and Technology (CISST'98)*, 1998, pp. 154–160.

[188] van Dijk, S., *Genetic Algorithms for Map Labeling*, Ph.D. thesis, Utrecht University, Department of Computer Science, Nov. 2001.

[189] van Dijk, S., Thierens, D., and de Berg, M., “On the design of genetic algorithms for geographical applications,” *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’99)*, edited by W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Morgan Kaufmann, Jul. 1999, pp. 188–195.

[190] van Wijk, J. J., van Liere, R., and Mulder, J. D., “Bringing computational steering to the user,” *Scientific Visualization - Dagstuhl’97*, IEEE Computer Society, 1997, pp. 304–313.

[191] Verner, O., Wainwright, R., and Schoenefeld, D., “Placing text labels on maps and diagrams using genetic algorithms with masking,” *INFORMS Journal on Computing*, Vol. 9, No. 3, 1997, pp. 266–275.

[192] Verweij, B. and Aardal, K., “An optimisation algorithm for maximum independent set with applications in map labelling,” *Proc. 7th Annu. Europ. Symp. on Algorithms (ESA’99)*, Vol. 1643 of *Lecture Notes in Computer Science*, Springer-Verlag, Prague, 16–18 Jul. 1999, pp. 426–437.

[193] von Laszewski, G., “Intelligent structural operators for the k-way graph partitioning problem,” *Proceedings of the 4th International Conference on Genetic Algorithms*, edited by L. B. Belew, Richard K.; Booker, Morgan Kaufmann, San Diego, CA, Jul. 1991, pp. 45–52.

[194] Wang, X., “Learning planning operators by observation and practice,” *Artificial Intelligence Planning Systems*, 1994, pp. 335–340.

[195] Ware, C., *Information Visualization: Perception for Design*, Morgan Kaufmann Interactive Technologies Series, 2000.

[196] Weitzman, L., “Designer: A knowledge-based graphic design assistant,” *Artificial Intelligence in Engineering Design*, edited by C. T. C. and D. Sriram, Vol. Volume 1. Design Representation and Models of Routine Design, Academic Press, San Diego, CA, 1992, pp. 433–466.

[197] White, G. M. and Wong, S. K. S., “Interactive timetabling in universities,” *Computers and Education*, Vol. 12, 1988, pp. 521–529.

[198] Wolff, A. and Strijk, T., “The Map-Labeling Bibliography,” `www.math-inf.uni-greifswald.de/map-labeling/bibliography`, 1996.

[199] Wong, K. H. and Ng, W. Y., “An interactive timetabling support system,” *Proc. of International Conference in System Management’90*, 1990, pp. 307–313.

[200] Yamamoto, M., Câmara, G., and Lorena, L. A. N., “Tabu search heuristic for point-feature cartographic label placement,” *GeoInformatica*, Vol. 6, No. 1, 2002, pp. 77–90.

[201] Yoeli, P., “The logic of automated map lettering,” *Cardographic J.*, Vol. 9, 1972, pp. 99–108.

[202] Zoraster, S., “Integer programming applied to the map label placement problem,” *Cartographica*, Vol. 23, No. 3, 1986, pp. 16–27.

[203] Zoraster, S., “The solution of large 0-1 integer programming problems encountered in automated cartography,” *Operations Research*, Vol. 38, No. 5, 1990, pp. 752–759.

[204] Zoraster, S., “Expert systems and the map label placement problem,” *Cartographica*, Vol. 28, No. 1, 1991, pp. 1–9.

APPENDIX **A**

Optimization Table

This appendix describes two tables that we built for experimenting with interactive optimization processes. The tables are based on the *optimization table* [8] developed by Joe Marks and Neal Lesh at the Mitsubishi Electric Research Laboratories (MERL) in Boston. MERL's table basically involves a computer, a whiteboard laid on a desk, and a data projector and a front-surface mirror[1], mounted on the ceiling. See Figure A.1 for illustration. The projector creates the image of the computer screen, which is reflected by the mirror to the whiteboard. The main input devices are a wireless mouse, a wireless keyboard, and a Mimio device[2] for controlling the mouse pointer. A mouse ring was also tried, but was not as effective for interaction as the previous interfaces. In addition, ordinary whiteboard pens and eraser can be employed to write notes on the table. This setup was used in several interactive optimization tasks [8, 124, 170].

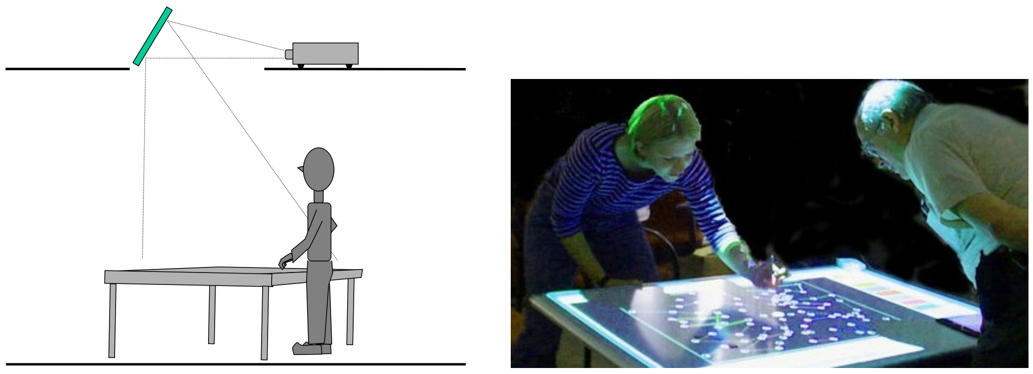


Figure A.1: MERL's Optimization Table. The photograph is courtesy of Joe Marks.

[1]A front-surface mirror is usually a metal plate with one side polished to function as a mirror. Ordinary glass mirrors are not suitable for projection because they produce a "ghost image".

[2]The Mimio device provides a solution for digitalizing information written on a whiteboard. It includes four special pens, an eraser, and a *Mimio bar* that is usually attached to the board. The pens and the eraser send a signal to the Mimio bar indicating which tool is being used and its current location. This signal is sent by the bar to the computer for recording the information written on whiteboard, or for controlling a mouse pointer. For the later case, the computer desktop image is projected on the whiteboard, and the X, Y-coordinates of the capture system of the Mimio bar is calibrated to match the projection area. More information about the Mimio device is available at *http://www.mimio.com*.

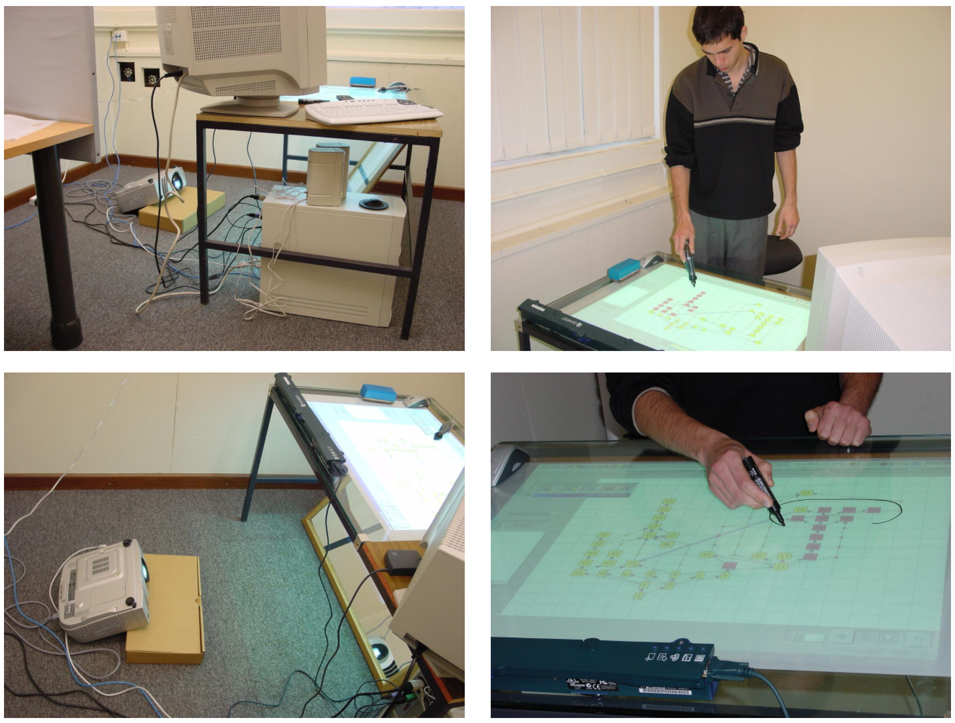


Figure A.2: Our first optimization table with bottom-up projection.

After a visit to MERL in the December 2000, we decided to build a similar environment, but having the computer screen projected under the table instead of on the top. The aim was to avoid shadow when the user is close to the table or is interacting with it.

Our first implementation (showed in Figure A.2) consisted of a normal desk modified to have a projection area. We used two layers of 6mm glass, with a semitransparent material in between. The image was created by a data projector (a Sony VPL-PX31, with 2800 lumens) standing on the floor, and was reflected to the bottom of the projection area by a large mirror under the table. The desk also had an area for keeping a monitor and additional tools. We installed a Matrox G450 video card with *dualhead* capability. The outputs of the video card were attached to the monitor and to the data projector, so that applications could be swapped between both screens. The interaction devices were a wireless mouse and a wireless keyboard, a Mimio device, and normal whiteboard pens and eraser, as in MERL.

We used this table in our initial tests of the GDHints system (before performing the pilot study). The possibility of directly interacting with the application and the large screen area showed to be fascinating and more intuitive than using a monitor. Nevertheless, we found some design problems: the projection took most of the space under the glass screen, so that it was not possible to sit

Equipments (acquired in 2001)	**Cost (AU$)**
Table (basic structure with glass layers) and platform – carpenter service	$4,500.00
1.5m x 1m Rear Light Projection Screen material, from *Screen Technics* (Australian company)	$110.00
Pentium IV system (1.4GHz, 512Mb, 40Gb HD, CD-ROM, soundcard, speakers, and 17" monitor) with a Matrox G450 video card	$2,700.00
Logitech wireless keyboard and mouse	$200.00
QTQ060 Mimio Capture Bar for PC	$1,900.00
VPL-PX31-2800 ANSI Sony Projector	$15,400.00
Total	**$24,810.00**

Table A.1: Cost of the equipments for the second optimization table.

properly, with the keens under the table; the environment was also too small for having more than one person working on it. Moreover, we had difficulty in adjusting the focus of the top and bottom edges of the image (due to a distortion of the lens of the projector). We minimized this effect by reversing the projector upsidedown; however, an optimal solution would demand a more elaborated positioning of the projector and the mirror. We then decided to build another table to solve some of these issues.

The second table is larger and taller than the first one – 157cm x 90cm by 90cm of height[3], with a projection area of 108cm x 80cm. It is placed on the top of a platform that we built, which floor is 125cm above the ground; the platform allows a sufficient distance between the data projector, that stays on the ground, and the glass screen of the table[4]. See Figure A.3. The image is projected straight up to the glass screen through a hole on the floor of the platform. This configuration permits up to four people discussing an optimization problem around the table. The computer and the input devices from the first environment were employed in the second table.

The new table was constructed for supporting a movable setup, by working also with a plasma screen. Unfortunately, our experiments with a Sony PFM-42B1 plasma 42" screen was not promising: this screen has a protective glass layer that stays about 2cm from the active display; such gap makes it difficult to interact directly with the image. Moreover, we noticed that the Mimio device had problems in detecting the Mimio pens when used on the plasma screen.

The total cost of the second optimization table is presented in Table A.1.

We did not perform any formal evaluation of the effectiveness of these two environments. One could design experiments, for example, to investigate whether users can produce better results in an optimization task by using the optimization tables rather than a normal monitor.

[3]Extensions to the legs of the table allow to incline it forwards.

[4]Another alternative would be to buy short-distance lens for the projector, but the cost-benefit for implementing this option was not advantageous.

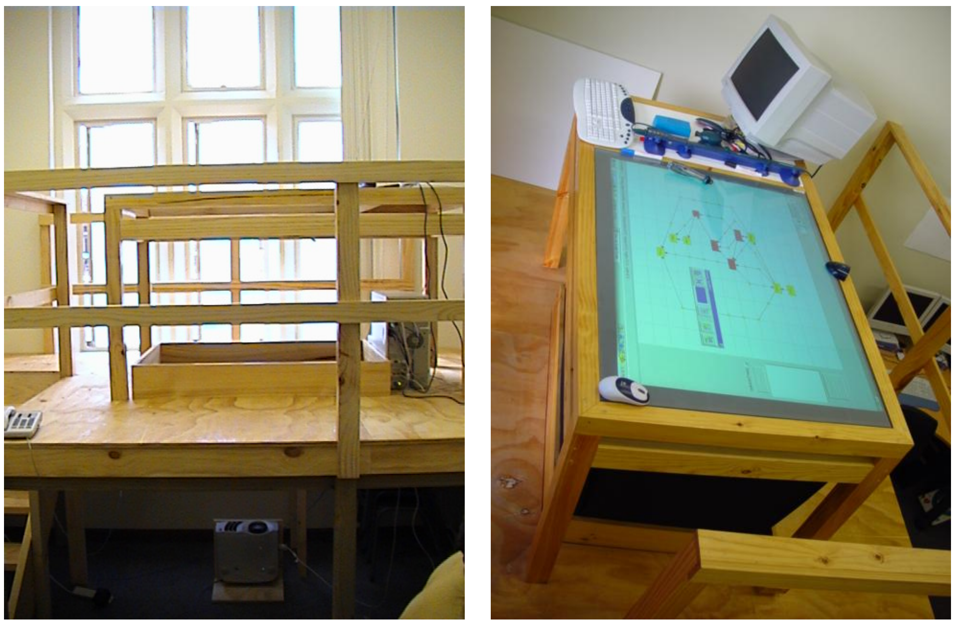


Figure A.3: Our second optimization table with bottom-up projection.

Nonetheless, we used the second table to demonstrate our interactive systems to professionals from Information Technology, Cartography, and Transport and Logistics Management, and we collected informal feedback from them. This feedback, together with our own observations were:

- The users mentioned feeling "involved" by the environment and being more concentrated on the problem. In fact, the size of the screen area takes a larger proportion of the field of vision of the user than a 17" monitor in a normal configuration.
- The experts in Cartography (from *Sydway*[5]) liked the table, and said that it would be interesting to have a similar setup at their work. Note that these professionals already use large workbenches, which are not computational except for a digitalizer (for inputting data).
- The Mimio device allows more intuitive forms of manipulating the elements of the interface than a mouse. It is possible, for instance, to have a tool in the GDHints system that allows the user to route edges by manually drawing them as lines.

The main disadvantages of the tables, observed during our experimentations, were:

- The glass surface reflects the light from the ceiling, which can by quite annoying. This effect was reduced by switching off the top lights when using the environment.

[5]Information about Sydway can be obtained from *http://www.ausway.com.au/*.

- The image resolution of most data projectors are considerably worse than the resolutions of good monitors. Moreover, computer monitors are more affordable than data projectors.
- The Mimio device had several problems: it is not very accurate, so clicking on small icons in the screen can be difficult; objects left on the table can block the signal sent by the Mimio pen; and the device does not implement hardware support for right-click.
- The Mimio also does not support more than one person controlling the computer simultaneously. There are other input devices for multiple-user applications. An example is the Diamondtouch technology [44], developed by MERL.
- Since the projected image is large, the user has to make wide movements with the arm to interact with it. This may be more tiring than using a mouse. We tried to minimize such problems by allowing the user to move the toolbars of the GDHints and LabelHints systems to the bottom of the screen.
- Finally, we still do not have a suitable configuration for sitting comfortably. The original table from MERL offers a better solution for this problem, but at a cost of shadows on the image. Mounting a table similar to the workbenches used by architects could be a good alternative.

Most of these disadvantages can be solved by improvements of touch screens, such as the ones used in the Tablet PCs. As this technology becomes more portable and affordable, large optimization tables may move from the research field to real applications in industry.

APPENDIX **B**

Included CD-ROM

This appendix describes the contents of the CD-ROM that comes with the thesis, and provides more details about the interface of the interactive systems described in the previous chapters.

B.1 Contents

The CD-ROM[1] is organized as follows:

- *Index.html* – a HTML file with the description of the contents of the CD-ROM, and with the terms and conditions for using this material.
- *UserHints.pdf* – this thesis in PDF format. A version for two-sided printing is available as *UserHints_2sided.pdf*.
- *ClusterHints* – the directory with the ClusterHints system.
- *GDHints* – a directory with versions of the GDHints system. This includes:
  - *FirstStudy* – the system and documents used in the first experiment.
  - *SecondStudy* – the system and documents used in the second experiment.
  - *FullVersion* – a complete version of the GDHints system.
- *LabelHints* – a directory with the LabelHints system.

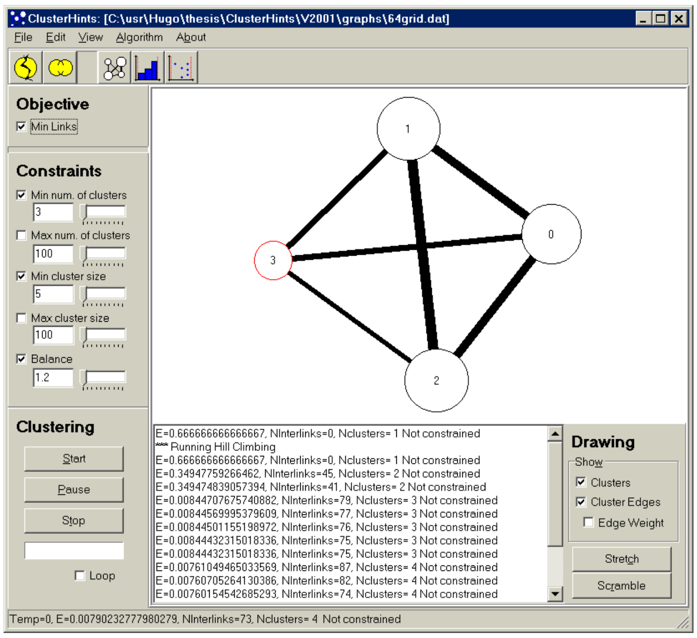
All programs in the CD-ROM are for Pentium PC with Microsoft Windows Me. They may run in other versions of Microsoft Windows (and even on Linux using Wine), but this is not guaran-

[1]This material is also available at the web site of the Information Visualization Research Group at the University of Sydney (*http://www.it.usyd.edu.au/˜visual*).

teed. Note that the programs are prototypes developed for testing our ideas only; their code is not optimized.

Next we describe the interface of the main programs included in the CD-ROM.

B.2 The ClusterHints System


ClusterHints: [C:\usr\Hugo\thesis\ClusterHints\V2001\graphs\64grid.dat]
File
Edit
View
Algorithm
About
1
0
3
2
Start
Pause
Stop
Stretch
Scramble

Objective
- [✓] Min Links

Constraints
- [✓] Min num. of clusters: 3
- [ ] Max num. of clusters: 100
- [✓] Min cluster size: 5
- [ ] Max cluster size: 100
- [✓] Balance: 1.2

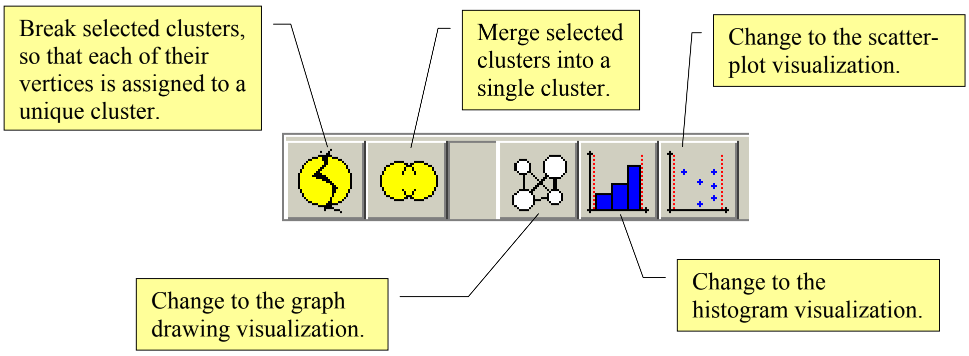
Clustering
- [ ] Loop

Drawing
Show
- [✓] Clusters
- [✓] Cluster Edges
- [ ] Edge Weight

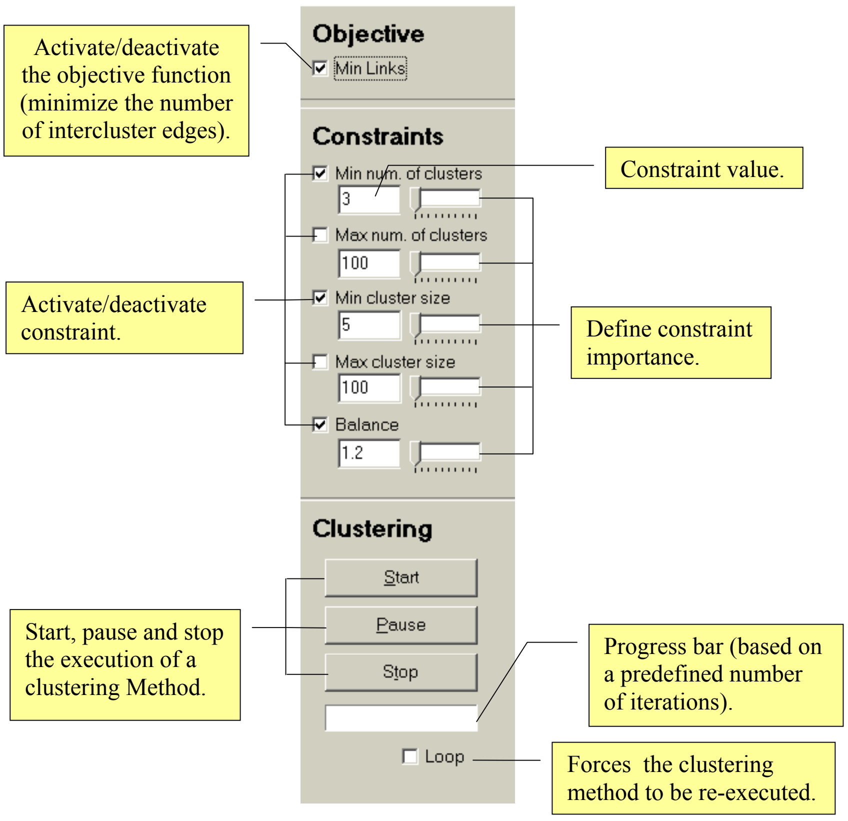
```
E=0.666666666666667, NInterlinks=0, Nclusters= 1 Not constrained
*** Running Hill Climbing
E=0.666666666666667, NInterlinks=0, Nclusters= 1 Not constrained
E=0.34947759266462, NInterlinks=45, Nclusters= 2 Not constrained
E=0.349474839057394, NInterlinks=41, Nclusters= 2 Not constrained
E=0.0084470767574 0882, NInterlinks=79, Nclusters= 3 Not constrained
E=0.00844569995379609, NInterlinks=77, Nclusters= 3 Not constrained
E=0.00844501155198972, NInterlinks=76, Nclusters= 3 Not constrained
E=0.00844432315018336, NInterlinks=75, Nclusters= 3 Not constrained
E=0.00844432315018336, NInterlinks=75, Nclusters= 3 Not constrained
E=0.00761049465033569, NInterlinks=87, Nclusters= 4 Not constrained
E=0.00760705264130386, NInterlinks=82, Nclusters= 4 Not constrained
E=0.00760154542685293, NInterlinks=74, Nclusters= 4 Not constrained
```

```
Temp=0, E=0.00790232777980279, NInterlinks=73, Nclusters= 4  Not constrained
```

Toolbar


Break selected clusters, so that each of their vertices is assigned to a unique cluster.
Merge selected clusters into a single cluster.
Change to the scatter-plot visualization.
Change to the graph drawing visualization.
Change to the histogram visualization.

Control Panel



Activate/deactivate the objective function (minimize the number of intercluster edges).

Constraint value.

Activate/deactivate constraint.

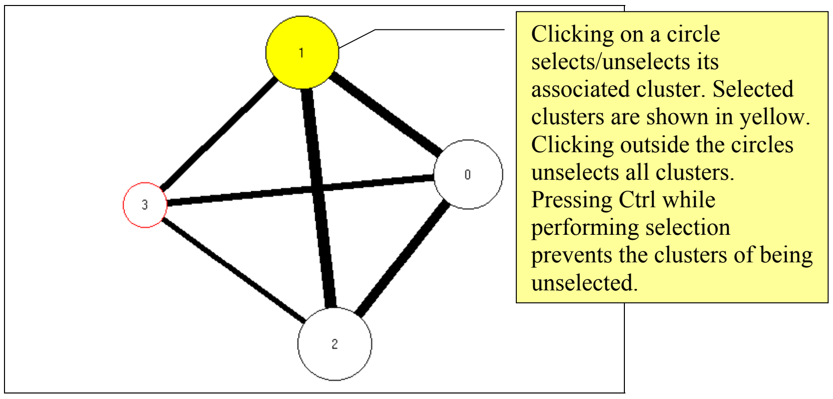
Define constraint importance.

Start, pause and stop the execution of a clustering Method.

Progress bar (based on a predefined number of iterations).

Forces the clustering method to be re-executed.

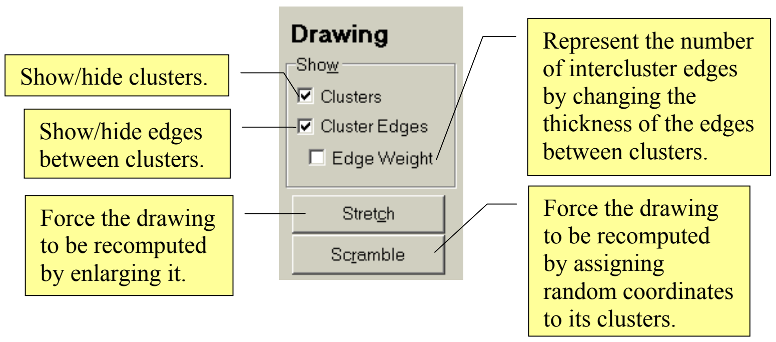
Interaction with the Graph Drawing visualization



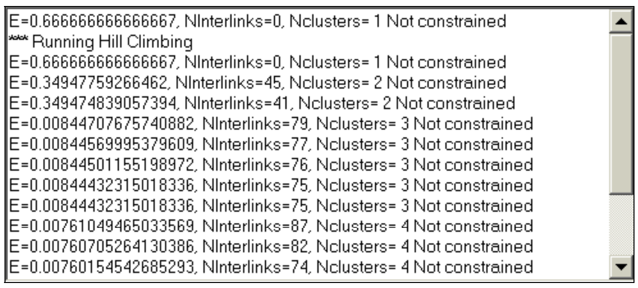
Clicking on a circle selects/unselects its associated cluster. Selected clusters are shown in yellow. Clicking outside the circles unselects all clusters. Pressing Ctrl while performing selection prevents the clusters of being unselected.

Direct manipulation is supported in this visualization for selection of clusters. The other visualizations do not implement interactive features.

Drawing Panel


Show/hide clusters.
Show/hide edges between clusters.
Force the drawing to be recomputed by enlarging it.
Drawing
Show
Clusters
Cluster Edges
Edge Weight
Stretch
Scramble
Represent the number of intercluster edges by changing the thickness of the edges between clusters.
Force the drawing to be recomputed by assigning random coordinates to its clusters.

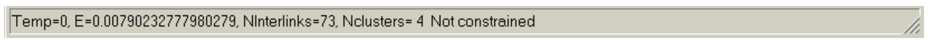
History



```
E=0.666666666666667, NInterlinks=0, Nclusters= 1 Not constrained
*** Running Hill Climbing
E=0.666666666666667, NInterlinks=0, Nclusters= 1 Not constrained
E=0.34947759266462, NInterlinks=45, Nclusters= 2 Not constrained
E=0.349474839057394, NInterlinks=41, Nclusters= 2 Not constrained
E=0.00844707675740882, NInterlinks=79, Nclusters= 3 Not constrained
E=0.00844569995379609, NInterlinks=77, Nclusters= 3 Not constrained
E=0.00844501155198972, NInterlinks=76, Nclusters= 3 Not constrained
E=0.00844432315018336, NInterlinks=75, Nclusters= 3 Not constrained
E=0.00844432315018336, NInterlinks=75, Nclusters= 3 Not constrained
E=0.00761049465033569, NInterlinks=87, Nclusters= 4 Not constrained
E=0.00760705264130386, NInterlinks=82, Nclusters= 4 Not constrained
E=0.00760154542685293, NInterlinks=74, Nclusters= 4 Not constrained
```

This area shows the quality of the solutions that were created by either the user or the system, and were displayed on the screen.

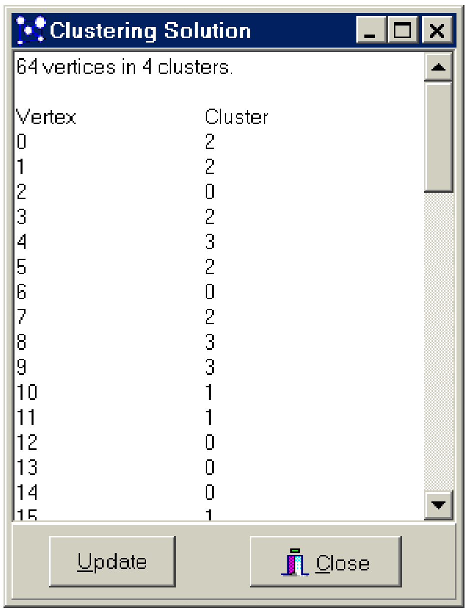
Status Bar



```
Temp=0, E=0.00790232777980279, NInterlinks=73, Nclusters= 4 Not constrained
```

The status bar presents the quality of the most recent solution created by the user or the system. When a clustering algorithm is running, this quality information refers to the intermediate solution being improved. If the algorithm is the Simulated Annealing, then the value of the temperature T is also showed.

Textual Description



```
64 vertices in 4 clusters.

Vertex          Cluster
0               2
1               2
2               0
3               2
4               3
5               2
6               0
7               2
8               3
9               3
10              1
11              1
12              0
13              0
14              0
15              1
```

This window presents the clustering solution textually. For each vertex v of the graph it shows the cluster id of v (the cluster id is an integer that uniquely identifies the cluster. These numbers are displayed at the center of the circles in the graph drawing visualization, and in the scatter-plot visualization).

Menu

File

Open – open a file containing the structure of a graph.

Exit – close the ClusterHints application.

Edit

Break Clusters – break selected clusters so that the resultant clusters have a single vertex each.

Merge Clusters – merge selected clusters into a single cluster.

Select All – select all clusters.

Unselect All – deselect all clusters.

Clear History – clear the history log.

View

Graph Drawing – change to the graph drawing visualization.

Histogram by Size – change to the histogram visualization, which shows the number of clusters by cluster size.

Clusters by Size – change to the scatter-plot visualization, which presents all clusters classified by cluster size.

Textual Description – show the Textual Description window.

Algorithm

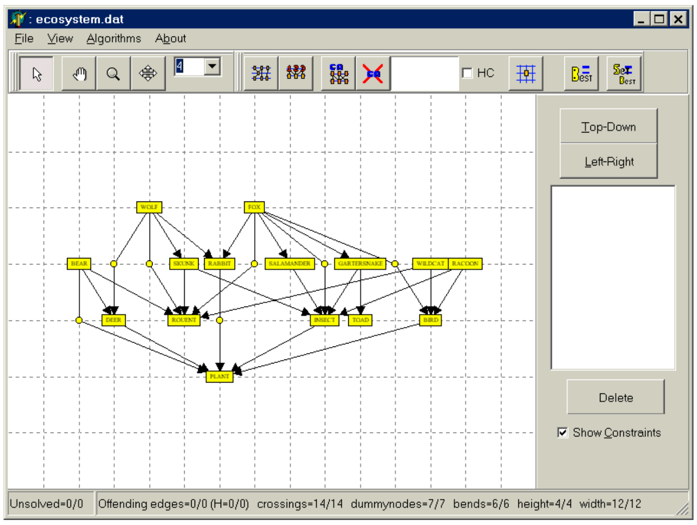
Hill Climbing – choose the Hill Climbing algorithm for graph clustering.

Simulated Annealing – choose the Simulated Annealing algorithm.

Lock Settings at Runtime – disallow changes of the constraint set and the objective function while an algorithm is executing.

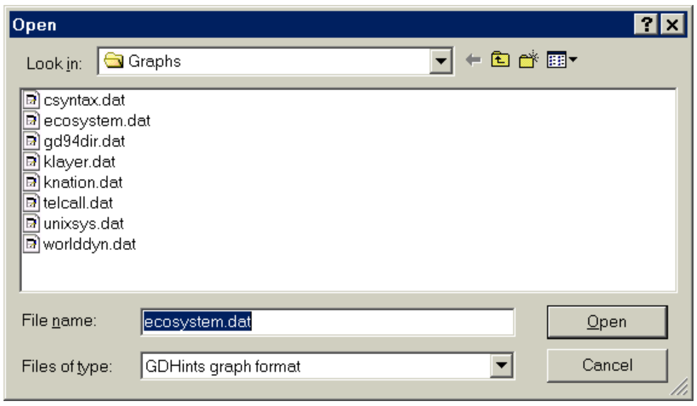
About – show information about the ClusterHints system.

B.3 The GDHints System – Full Version



Unsolved=0/0 Offending edges=0/0 (H=0/0) crossings=14/14 dummynodes=7/7 bends=6/6 height=4/4 width=12/12

Opening a Graph



csyntax.dat
ecosystem.dat
gd94dir.dat
klayer.dat
knation.dat
telcall.dat
unixsys.dat
worlddyn.dat

File name: ecosystem.dat
Files of type: GDHints graph format

In order to open a graph the user must click on *File—Open* and choose a graph file in the *Open Dialog window* (all graph files have extension ".dat"). Then he or she must press *Open*. The system automatically produces an initial drawing for the graph by selecting all vertices and running the Sugiyama method.

After loading the graph structure, it is possible to open a saved drawing for it (if there is one). In that case, the user goes to the *Open Dialog window* again, changes the field *file type* to "GDHints drawing", and opens a file with extension ".drw". The drawing is recovered if it is associated with the current graph in memory.

The graph file of the GDHints system is a text file that contains:

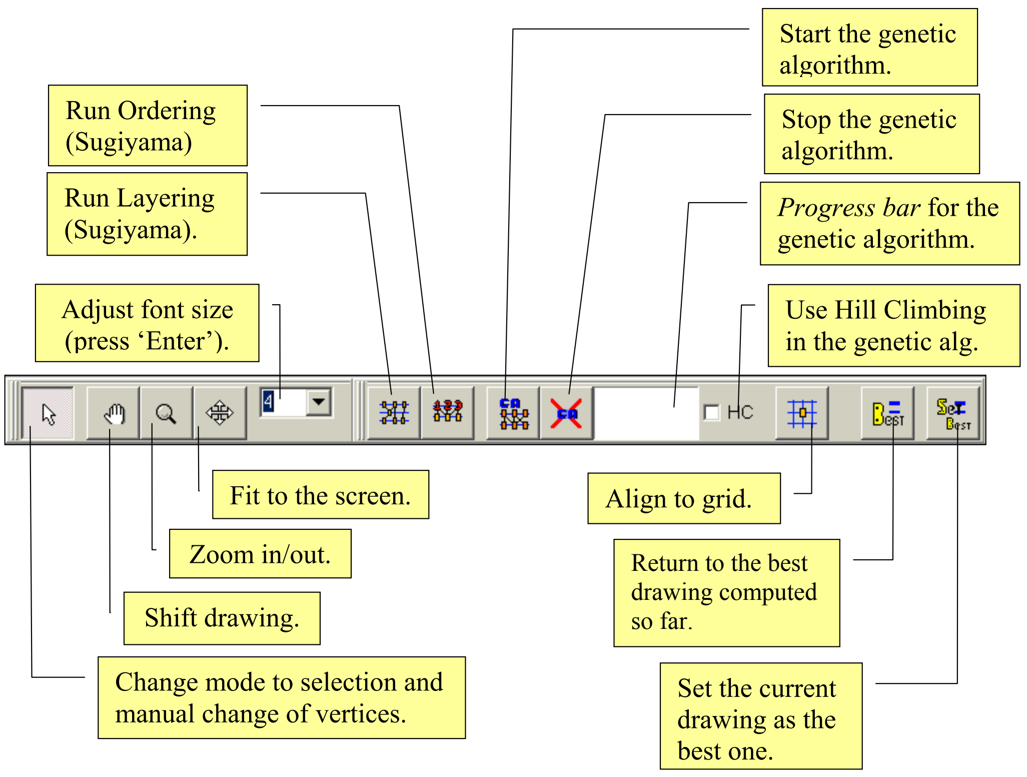
1. A line with a general description.

2. A line with two integers, n and m (representing the number of vertices and the number of edges of the graph, respectively), separated by a space.

3. An empty line or a line with an ordered list of labels for the vertices, separated by spaces. This line should have at most n labels.

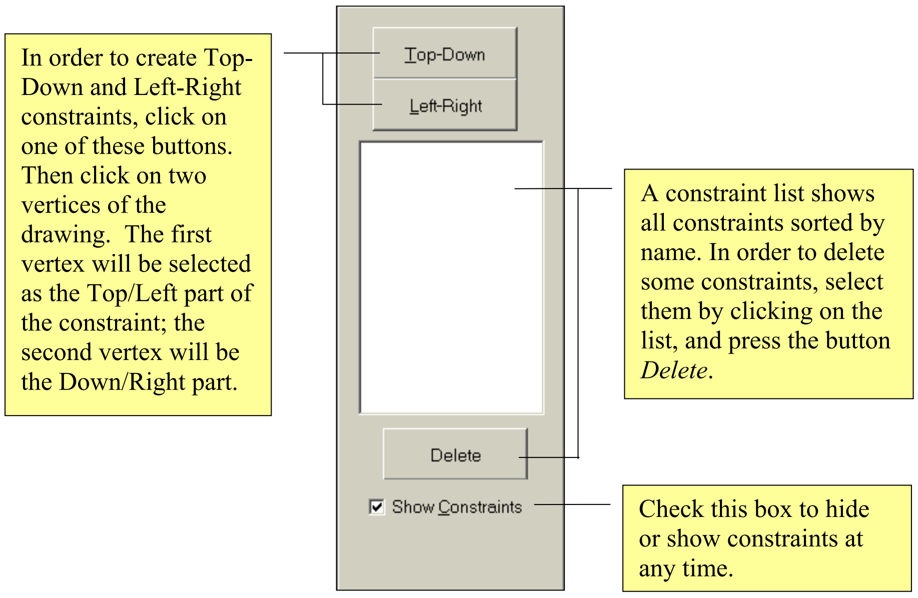
4. One or more lines with m edges in the form "(u, v)", where u and v are indices of the vertices, $1 \leq u, v \leq n$. The edges can be separated by a space, a tab, or by the "new line" code.

Toolbar


Start the genetic algorithm.
Run Ordering (Sugiyama)
Stop the genetic algorithm.
Run Layering (Sugiyama).
Progress bar for the genetic algorithm.
Adjust font size (press 'Enter').
Use Hill Climbing in the genetic alg.
4
HC
Fit to the screen.
Align to grid.
Zoom in/out.
Return to the best drawing computed so far.
Shift drawing.
Change mode to selection and manual change of vertices.
Set the current drawing as the best one.

Note that the Layering, the Ordering and the genetic algorithm only run on selected vertices. If no vertex is select, then no action is performed by these algorithms. The tool for aligning vertices to the grid also works only on selected vertices.

Constraint Panel


In order to create Top-Down and Left-Right constraints, click on one of these buttons. Then click on two vertices of the drawing. The first vertex will be selected as the Top/Left part of the constraint; the second vertex will be the Down/Right part.

A constraint list shows all constraints sorted by name. In order to delete some constraints, select them by clicking on the list, and press the button *Delete*.

Check this box to hide or show constraints at any time.

Interaction with the visualization

The visualization of the GDHints system displays the working solution, which consists of a graph drawing. Vertices are shown as rectangles, and edges as lines. Direct manipulation is supported for:

- **Vertex Selection** – Clicking on a vertex selects it. Selected vertices are shown in red. In order to select several vertices simultaneously, the user has to press and hold the *Control* key while selecting. The *Shift* key allows the user to reverse the current selection of a vertex. Furthermore, it is possible to select a group of vertices by using the mouse to define a "selection area".

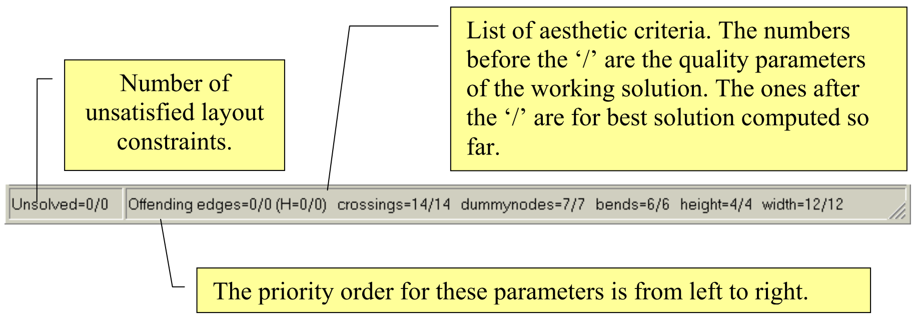
- **Manual Changes** – Ideally, vertices should be positioned on the intersection points of the grid. However, the system allows the user to use the intermediate X-coordinates.

- **Constraint Insertion** – See explanation in the previous section.

Best Drawing

The system is capable of recognizing when a new drawing generated by the user or by the drawing algorithm is better than the best drawing produced so far. When this occurs the system flashes the button *Best* () and saves the new drawing as the best one. The user can also force the current solution to be set as the best drawing at any time by pressing the button *Set Best* (). This operation can be executed even when the current solution is worse than the best drawing held by the system. The user can recover the best drawing by clicking on *Best* ( ).

Status Bar


Number of unsatisfied layout constraints.
List of aesthetic criteria. The numbers before the '/' are the quality parameters of the working solution. The ones after the '/' are for best solution computed so far.
Unsolved=0/0
Offending edges=0/0 (H=0/0) crossings=14/14 dummynodes=7/7 bends=6/6 height=4/4 width=12/12
The priority order for these parameters is from left to right.

The status bar presents the quality parameters of the working solution and of the best solution.

Menu

File

Open – open a graph or a drawing of a graph.

Save – save a graph or a drawing.

Print – print the drawing.

Voice – activate/deactivate the voice feedback of the best solution agent.

Exit – close the GDHints application.

View

Layers – show/hide layers (grid rows).

Columns – show/hide grid columns.

Dummy Vertices – show/hide dummy vertices.

Constraints – show/hide constraints.

Fit to Screen – fit the drawing to the screen area.

Always Fit to Screen – force the system to automatically fit the drawing to the screen area after an update of the work solution.

Font Size Adjustment – set the font size adjustment as *automatic* or *manual*.

Toolbars – show/hide the toolbars and the constraint panel.

Algorithms

Layering – run the layering algorithm based on the Sugiyama method.

Ordering – run the ordering algorithm based on the Sugiyama method.

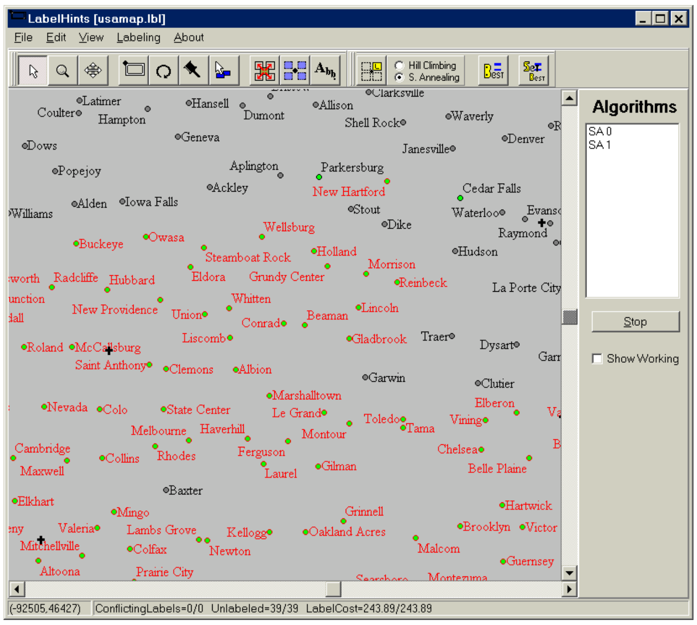
Start GA – start the genetic algorithm.

Stop GA – stop the genetic algorithm.

Align Vertices to Grid – move all selected vertices to the nearest intersection position of the grid.

About – show information about the GDHints system.

B.4 The LabelHints System

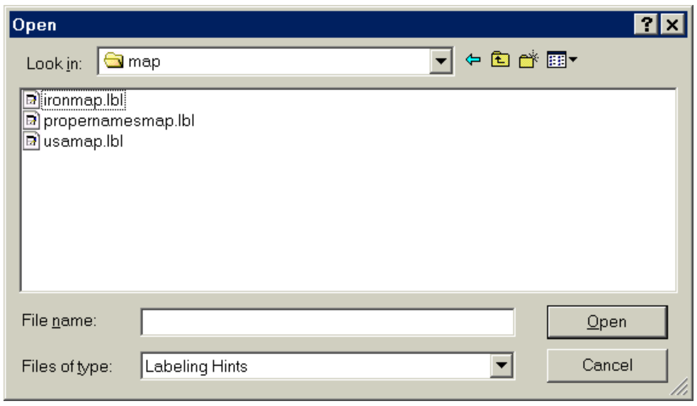


(-92505,46427) ConflictingLabels=0/0 Unlabeled=39/39 LabelCost=243.89/243.89

Opening or Creating a Labeling Problem

A labeling process starts by opening an existing labeling problem from the disk, or by creating a labeling problem manually using the graphical interface.

In order to open a problem, the user must choose the option *File—Open* of the menu, and then select the desired file (with extension '.lbl') in the *Open Dialog windows*:



Look in: map

ironmap.lbl
propernamesmap.lbl
usamap.lbl

File name:

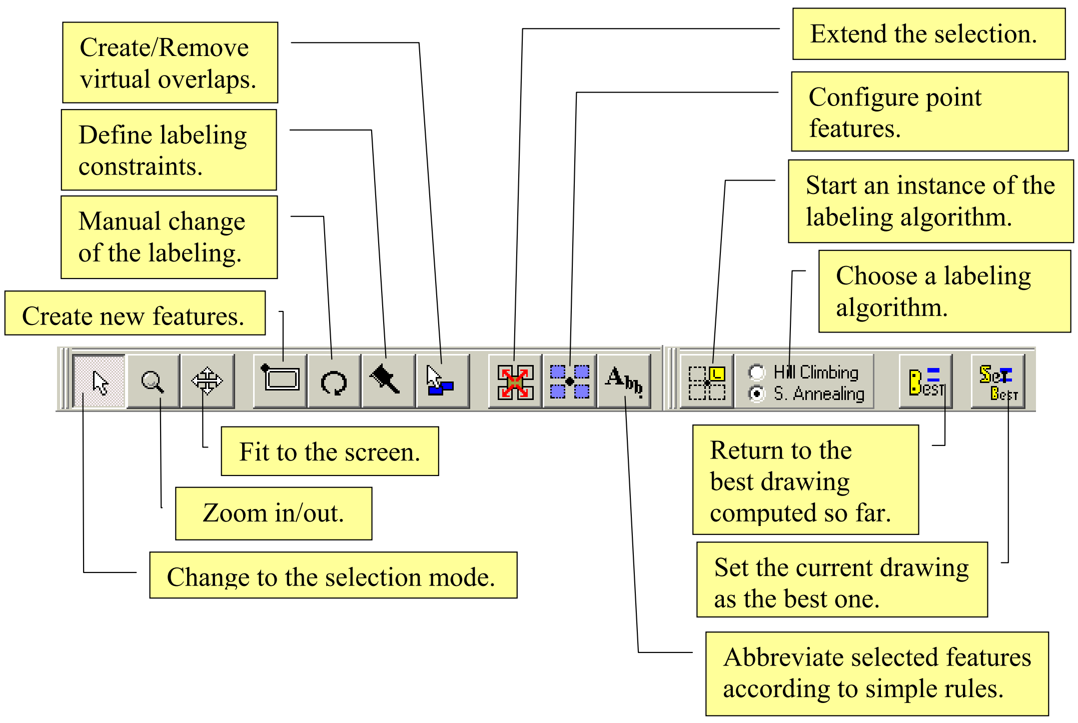
Files of type: Labeling Hints

The LabelHints system comes with three labeling samples: a map with iron mines in Tasmania ('ironmap.lbl'), a randomly-generated map with proper names ('propernamesmap.lbl'), and a map of cities in Iowa ('usamap.lbl'). These maps are only for testing our prototype. They should not be used for any other reason.

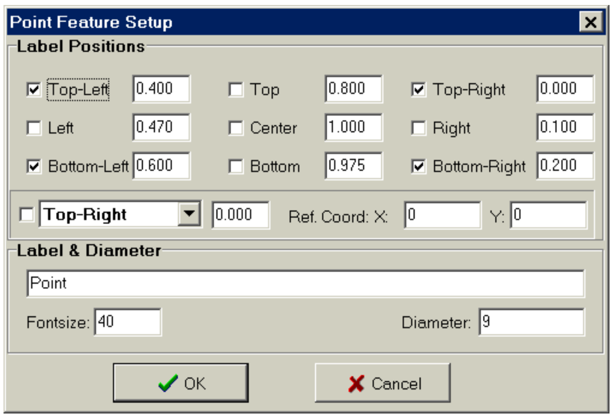
After choosing the labeling file, the system opens and presents the map. Maps with many features may take a while to be drawn on the screen.

Another possibility is to create a new map. In order to do this, the user must define the general properties of point features by clicking on the *Feature Setup* button (). A dialog box appears for configuring candidate positions, the font size, and the diameter of the features' points. The next step is to select the button for creating new features (), and clicking on the desired positions of the screen. The feature attributes (such as the label positions) can be changed later by selecting one or more features, and clicking on the Feature Setup button again.

Toolbar


Create/Remove virtual overlaps.
Define labeling constraints.
Manual change of the labeling.
Create new features.
Extend the selection.
Configure point features.
Start an instance of the labeling algorithm.
Choose a labeling algorithm.
Hill Climbing
S. Annealing
Fit to the screen.
Zoom in/out.
Change to the selection mode.
Return to the best drawing computed so far.
Set the current drawing as the best one.
Abbreviate selected features according to simple rules.

Feature Setup



Point Feature Setup

Label Positions

☑ Top-Left	0.400	☐ Top	0.800	☑ Top-Right	0.000
☐ Left	0.470	☐ Center	1.000	☐ Right	0.100
☑ Bottom-Left	0.600	☐ Bottom	0.975	☑ Bottom-Right	0.200

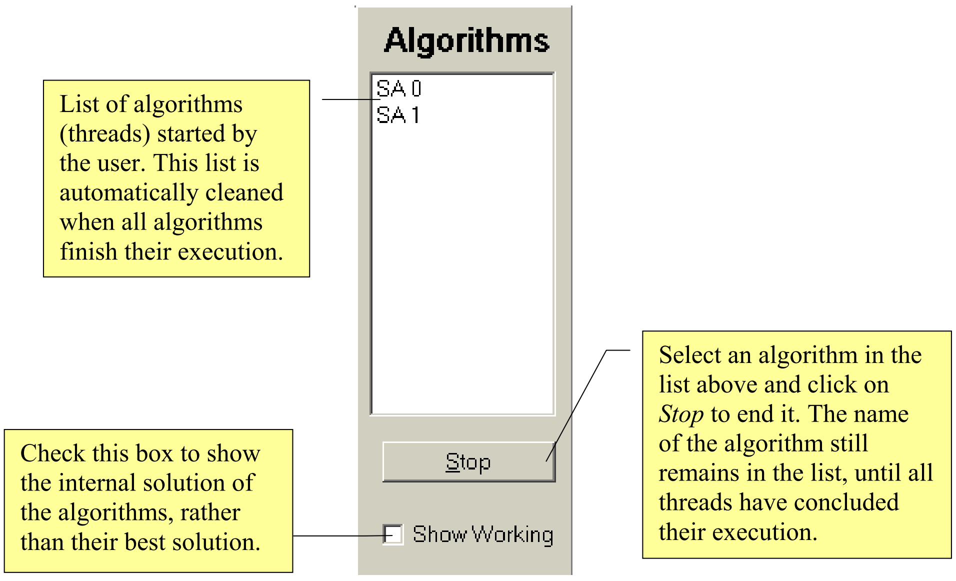
☐ Top-Right 0.000 Ref. Coord: X: 0 Y: 0

Label & Diameter

Point

Fontsize: 40 Diameter: 9

Algorithm Panel



List of algorithms (threads) started by the user. This list is automatically cleaned when all algorithms finish their execution.

Check this box to show the internal solution of the algorithms, rather than their best solution.

Select an algorithm in the list above and click on *Stop* to end it. The name of the algorithm still remains in the list, until all threads have concluded their execution.

Interaction with the visualizations

Two visualizations are implemented in the LabelHints system: a geographical map, and a drawing of the conflict graph. These visualizations are also described in the Chapter 7, Section 7.4.4. Unlabeled features are shown as crosses in the map visualization.

The user can interact with the visualizations for:

- **Feature Selection** – In the selection mode ( ), the user can click on the point (circle or cross) of a feature in order to select it. Several features can be simultaneously selected by pressing and holding the *Control* key while performing the selection. Holding the *Shift* key allows the user to reverse the current selection of a feature. Similarly to the GDHints system, it is possible to select a group of features by defining a “selection area” with the mouse. If the *Alt* key is pressed during the selection, then the Selection Extension mechanism is applied to the newly selected features.

- **Feature Information** – In the selection mode, the user must click on the point of a feature and wait some seconds in order to see the feature’s label.

- **Manual Changes** – In the manual change mode (), clicking on the point of a feature changes its associated label position in use. The left and the right buttons of the mouse change the label position in different directions. The middle button sets the feature as unlabeled. By using the left mouse button it is possible to drag and drop a label to any of its predefined candidate positions. The left button also allows a drag-and-drop operation, but it is used for free movements; a labels can be placed anywhere on the map, causing a customized candidate position to be created or updated.

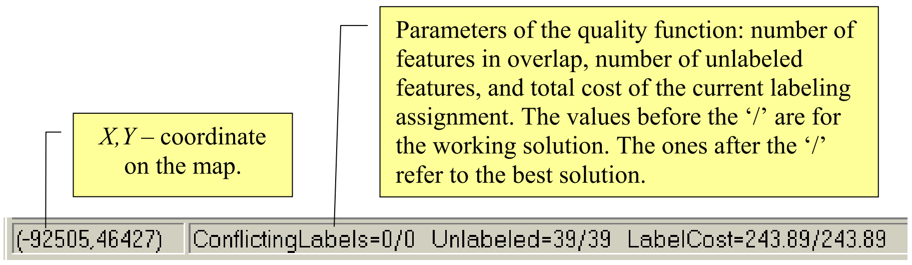
- **Labeling Constraint Insertion** – Labeling constraints can be defined by using the *Pin* tool (). Clicking with the *Pin* on a feature constrains it to be labeled. Clicking on a label prevents labeling algorithms to move it to another label position.

- **Virtual Overlap** – The virtual overlap tool () allows the user to create or remove a virtual overlap between two features, or between a feature and a label (belonging to another feature). This can be done by clicking on a pair of elements on the screen. Note that this tool can be used to remove original edges of the conflict graph, which represent real overlaps between candidate positions.

Best Labeling Solution

The LabelHints system implements the best solution agent as in the GDHints system. The buttons *Best* () and *Set Best* () allow the user to recover the best solution or to set the best solution respectively.

Status Bar



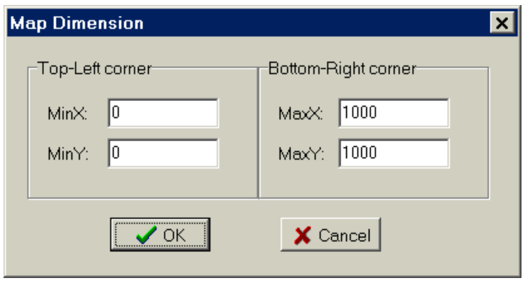
X, Y – coordinate on the map.

Parameters of the quality function: number of features in overlap, number of unlabeled features, and total cost of the current labeling assignment. The values before the '/' are for the working solution. The ones after the '/' refer to the best solution.

```
(-92505,46427)   ConflictingLabels=0/0  Unlabeled=39/39  LabelCost=243.89/243.89
```

The status bar presents the quality parameters of the working solution and of the best solution.

Map Dimension



Map Dimension

Top-Left corner

MinX: 0

MinY: 0

Bottom-Right corner

MaxX: 1000

MaxY: 1000

Menu

File

New – create a new labeling problem.

Open – open a labeling problem.

Save – save the current labeling problem.

Save As – save the current labeling problem with another name.

Print – print the labeling as it appears on the screen.

Voice Feedback – activate/deactivate the voice feedback of the best solution agent.

Exit – close the LabelHints application.

Edit

Map Dimension – define the maximum and minimum X and Y coordinates of the map. These coordinates are used for adjusting (fitting) the map to the screen area. This tool also offers the option of removing candidate positions and/or features that are out of the map dimension.

Point Feature Setup – open a dialog box for configuring the parameters of new point features (if now feature is selected); if some features are selected, then this dialog box allows the user to adjust their attributes.

Abbreviate Selected Labels – allows the user to abbreviate selected features. This is a simple facility that was implemented based on suggestions of the domain experts.

View

Display Labeling – switch to the map visualization.

Display Conflict Graph – switch to the conflict graph visualization.

Map Geometry – show/hide the geographic map in background.

Label Text – show/hide the labels.

Candidate Label Positions – show/hide all candidate positions.

Label Position in Use – show/hide the candidate position in use.

Highlight Overlap – highlight or not overlaps on the map.

Hide:

Unselected Unlabeled features

Unselected Additional Label Positions – show/hide candidate positions that not selected. This must be used in combination with the option *Candidate Label Positions*.

Toolbars – show/hide the toolbars and the algorithm panel. Also allows the user to move the toolbars to the bottom of the window.

Labeling

Hill Climbing – run the labeling Hill Climbing algorithm. This algorithms iterates while there is a change of the labeling that results in a better solution or in a solution with the same cost.

Simulated Annealing – run the Simulated Annealing algorithm.

Reset Labeling – set all selected feature to ‘unlabeled’.

About – show information about the LabelHints system.