Graphic Object Layout with Interactive Genetic Algorithms

Toshiyuki MASUI
SHARP Corporation
Information System R&D Center
Tenri, Nara 630, Japan

Abstract

Automatic graphic object layout methods have long been studied in many application areas in which graphic objects should be laid out to satisfy the constraints specific to each application. In those areas, carefully designed layout algorithms should be used to satisfy each application's constraints. However, those algorithms tend to be complicated and not reusable for other applications. Moreover, it is difficult to add each user's preferences to the layout scheme of the algorithm. To overcome these difficulties, we developed a general-purpose interactive graphic layout system GALAPAGOS based on genetic algorithms. GALAPAGOS is general-purpose because graphic objects are laid out not by specifying how to lay them out, but just by specifying the preferences for the layout. GALAPAGOS can not only lay out complicated graphs automatically, but also allow users to modify the constraints at run time so that users can tell the system their own preferences.

1 Introduction

How to lay out many graphic objects efficiently or attractively into a limited two-dimensional space is an important issue in applications like text formatters, data visualizers, LSI block layout tools, and many others. Since graphic layout is also important in interactive environments, graphic constraint solving mechanisms are becoming popular in user interface design tools [11] [12] [17]. A constraint like "the left edge of rectangle A should have the same X-coordinate as rectangle B" always makes the two rectangles move together, and users do not have to move rectangle A when they moved rectangle B. This kind of constraint solving system greatly helps users arrange graphic objects interactively.

The difficulty of solving the constraints depends on the nature of the constraints. Linear constraints like the one shown above can be solved quickly enough to be used in interactive environments, but complicated constraints for neatly laying out many graphic objects are hard to solve, and there is not always a procedure to get the optimal solution. Heuristics have long been used for these kind of problems where getting the optimal solution are almost impossible. However, it is usually difficult to find good heuristics, and it can never be general-purpose and you may have to use completely different set of heuristics for solving only slightly different constraints.

Recently, stochastic algorithms like *Simulated Annealing* [8] and *Genetic Algorithms* (GA) [3] [6] are becoming widely used for graphic object layout problems. They do not try to solve the constraints directly, but they modify the candidate solutions with random values, and make them approach the optimal solution through many trials. Using these algorithms, you should just specify what kind of constraints you want to solve, and good solutions are calculated automatically.

Although stochastic methods are powerful and able to solve complicated constraints, they still have limitations. It is sometimes difficult for the users to get preferred results, since users cannot specify how to lay out objects. Also, some constraints are inherently hard to specify. It would not be easy to specify constraints which can make all kinds of graphs look neat.

To solve these problems, we developed a GA-based interactive graphic layout system GALAPAGOS (Genetic ALgorithm And Presentation-Assisted Graphic Object layout System.) GALAPAGOS not only works as an automatic graphic layout system, but allows users to modify the constraints at run time so that they can tell the system their preferences. When a user notices that the system is going to have a solution which he doesn't like, he can tell so to the system at any moment by adding or modifying the constraints currently used for the calculation, and get preferred result.

2 Genetic algorithms for graphic layout

In this section, we briefly introduce how genetic algorithms work, and show how they can be applied to graphic layout problems.

1049-2615/92 $3.00 ©1992 IEEE

2.1 Genetic algorithms overview

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. The solution domain of a problem is represented as a bit string (or a *gene*.) Starting from a randomly initialized set of genes, those genes evolve to better genes which are closer to the solutions of the problem by iterations of modifications to the genes. Each iteration is called a *generation*. The modification in a generation is performed like this. First, the "performances" of all of the genes in the set are calculated by a *gene evaluation function*. Second, genes with good performances are chosen as candidate genes for the next generation. Finally, *genetic operations* are performed to some portion of the selected genes so that better genes are produced. Genetic operations usually consist of *crossover* and *mutation*, simulating real genes. Crossover is performed by selecting two genes, cutting them at two randomly selected points, and exchanging the portion of the genes between the cut points. (See Figure 1.) Mutation is performed by flipping the value of a bit in a gene in some small fixed rate.

The whole procedure simulates evolution through natural selection. Although each gene has no knowledge about the problem to solve, good genes survive to the next generation and bad genes will die out. Two genes which have good feature at different portions of the genes will produce a better gene through crossover. Even better genes may be produced by mutation, and eventually dominate the population.

In spite of their simplicity, genetic algorithms are powerful and robust. Genetic algorithms are drawing many attentions recently and are becoming used more and more in various problem areas where extensive search is impossible or no good algorithm can be found to solve the problems. Recent application areas are listed in [3].

Many modifications to the algorithm have been proposed to improve the performance. Also, there are many parameters used in the algorithm which can change the behavior of the algorithm to a great extent. Some of them are the *population* (number of genes,) *mutation rate*, and *crossover rate* (ratio of genes used for crossover.) The evaluation function also plays an important roll in the calculation and determines the whole performance of the algorithm. Proper combination of parameters and the evaluation function should be carefully chosen to get good performances.

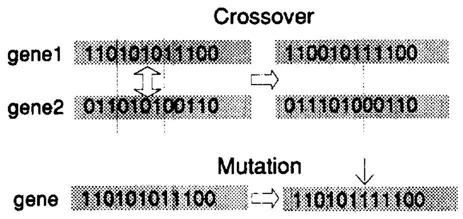


Figure 1: Genetic operations - crossover and mutation

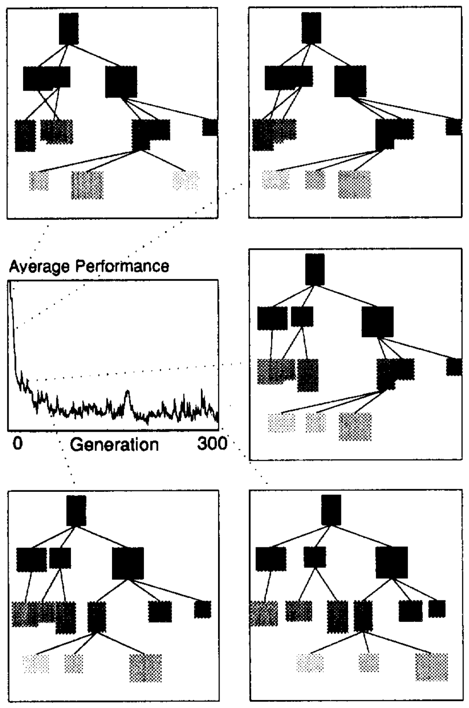


Figure 2: Tree nodes layout: best layouts at some of the generations

2.2 A simple example of using genetic algorithms for graphic layout

Figure 2 shows how genetic algorithms can be used to lay out a tree structure. The X-axis of the performance graph shown at the left-center corresponds to the generation, and the Y-axis shows the average performance of the genes used in the calculation [1].

Three criteria are used for this layout. They are 1) *nodes should not overlap*, 2) *children nodes should be correctly ordered*, and 3) *a parent node should be at the center of its children nodes*. These criteria are evaluated through an evaluation function shown below. In Figure 2, nodes with lighter color at the left show the correct node ordering.

[1]This is a typical shape of performance improvement through time.

```
int evaluategene(char *gene)
{
     int val = 0;
     Decode the node informations from  gene;
     for (all nodes) {
          if (the node has overlapped portion
               with its right node) {
                val += (overlapping width) * 100;
          }
     }
     for (all nodes) {
          val += sqr(parent position  -
                            center position of its children);
     }
     /*  other criteria  */
     return val;
}
```

In this example, even though we did not specify how to lay out the nodes to satisfy the criteria, the final solution satisfies all those criteria shown above and looks balanced. Another criterion like 4) *spaces between nodes should have the same width* can easily added to improve the appearance, just by adding a few more lines to **evaluategene()**.

Laying out tree nodes under the criteria like the ones shown above is not a hard problem, and many fast algorithms have been proposed. However, using genetic algorithms still has advantages, since they are robust to the change of the criteria.

3 GALAPAGOS overview

GALAPAGOS is a GA-based graphic layout system which allows interactive runtime modification of constraints defined among graphic objects. With a input data representation and constraints defined among the components, GALAPAGOS tries to lay out those components in a rectangle using genetic algorithms and displays the best solution. After looking at the displayed solution, users of GALAPAGOS can edit it with a graphic editor to modify or add new constraints. For example, when a user noticed that an object is placed in a wrong position, he can drag it to the right position and make it immovable. When a user noticed that two objects should have the same Y-coordinate to make the whole structure look neat, he can specify that one's Y-coordinate is the same as the Y-coordinate of the other. Also, he can modify the various parameters used in genetic algorithms. For example, he can modify the mutation rate at any time during the computation to get better results.

GALAPAGOS is based on the same algorithm as the GENESIS system[4]. The main loop of GALAPAGOS is shown below:

```
initialize();
for(gen=0; gen < maxgen; gen++){
     check_event();
     selectgene();
     mutate();
     crossover();
     evaluate();
     measure();
     display();
}
```

selectgene() selects genes from the population based on Baker's method[1]. With this method, the worst performance is defined, and genes whose performance is worse than that value will not be used in the next generation. Genes are selected so that the population of the gene after **selectgene()** is proportional to the difference between the worst value and the performance of the gene. The worst value is updated at each generation.

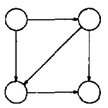
The only difference between the algorithms of GALAPAGOS and GENESIS is **check_event()**, where users can tell GALAPAGOS to stop and modify current constraints used in **evaluategene()**. Users can either continue the loop or start the loop again after the change.

After making modifications to the layout and constraints, the edited layout data is copied back to one third of the population so that the edited data will prevail in the population.

4 Using GALAPAGOS for directed graph layout

4.1 Directed graph layout problem

A *directed graph* is a graph which consists of a set of nodes N and a set of arcs E. An arc is an ordered pair of nodes (n, m), where n is called the *tail* and m is called the *head.* Below is a directed graph with four nodes and five arcs.



When a directed graph has many nodes and arcs, it becomes very hard to lay out all of them so that the graph look nice to humans. Many constraints can be defined to make the layout look nice. Some of them are listed in Figure 3.

Many graphic layout algorithms have been proposed to solve these or other constraints[16]. However, finding a layout which gives minimal number of line crossing is an NP-hard problem, and many other problems are, too. So, most of the algorithms for laying out directed graphs are using some heuristics.

4.2 Directed graph layout by GALAPAGOS

GALAPAGOS can not only automatically lay out a directed graph nicely, but also allow the user to edit the result

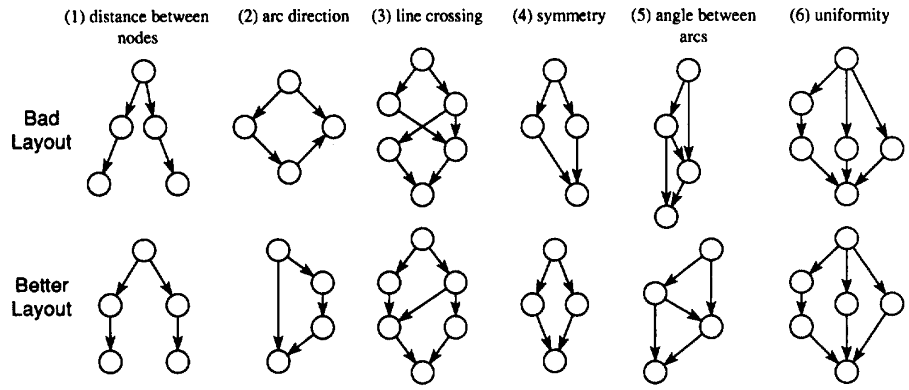


Figure 3: Constraints used in the layout of directed graphs

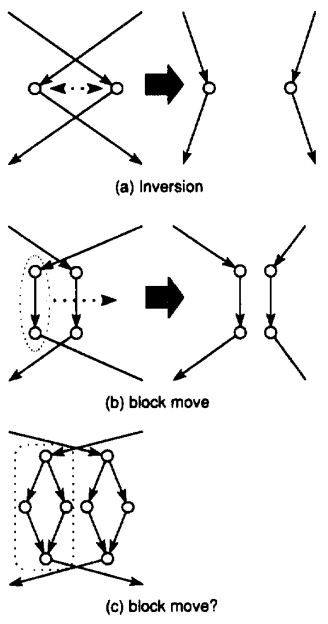


Figure 4: Inversion and block move

and add or modify the constraints so that the user can tell the system his preferences.

4.2.1 Modifications to standard genetic algorithms

Slight modifications are made to standard genetic algorithms shown before. First, "integer encoding" is used instead of bit string encoding. The positions of all the nodes are represented as an integer array, and the array is treated as a gene. Crossover is performed by exchanging a portion of two arrays. Mutation is performed by giving a random value to an array element. Second, *inversion* is used as another genetic operator. Inversion means exchanging the positions of two nodes. Figure 4(a) shows how inversion works for the layout of a directed graph. *Block move* shown in Figure 4(b) can be another candidate for a new GA operator. However, it does not work as good as inversion, because it does not usually improve the layout unless proper set of nodes are chosen as the block like shown in Figure 4(c). For this reason, it is not used as a genetic operator in GALAPAGOS.

4.2.2 Constraints used in the layout

Constraints shown below are used as default constraints.

1. The head of an arc should be below the tail.
2. The distance between every pair of nodes should be bigger than a constant value.
3. The number of arc crossings should be as small as possible.
4. The angle between two arcs should be bigger than a constant value.

Constant weight values are defined to each constraint, and for each violation to the constraints, the value is added to the return value of the evaluation function. In the examples shown below, (3000, 400, 300, 400) is used as the weight values. For example, 300 is added to the return value of the evaluation function for each line crossing.

In addition to them, constraints shown below can be added by the user interactively.

5. A node is positioned at a place specified by the user.
6. Two specified nodes have the same X-coordinate.
7. Two specified nodes have the same Y-coordinate.

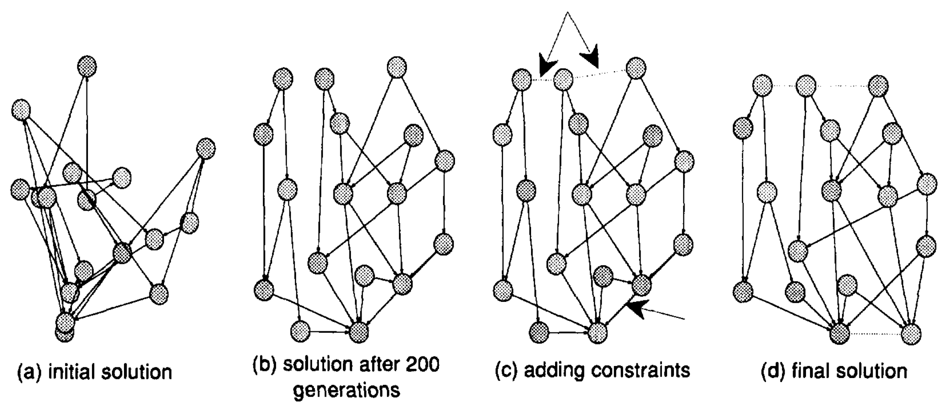


Figure 5: Sample layout session

4.2.3 Examples

In our current implementation, the program which executes genetic algorithms (called *GA Visualizer*) can display the best layout, but cannot get user inputs. So a commercial graphic editor is used for the modifications of the constraints. The initial data is given by the graphic editor or some other programs to GA Visualizer. GA Visualizer then performs genetic algorithms and displays the best result at each generation. If the user wants to modify the constraints, he edits the result using the graphic editor, and gives them back to GA Visualizer.

Figure 5 shows a sample layout session using the GALAPAGOS system. First, link information of a directed graph is given to the system. The system starts executing genetic algorithms, and displays the best solution at each generation. Figure 5(a) is the first solution found by the system. After many generations, much better solution (b) is found. At this point, the user notices that the graph looks better if the three nodes at the top and the two nodes at the bottom are aligned with the same Y-coordinate. So he puts three constraints to the graph shown by big arrows in (c), and continues calculation. New constraints are shown as dotted lines. He finally gets the final layout shown in (d).

Figure 6 shows another example. Here, the user edits the initial solution (a), and make three nodes immovable so that the numbers on the nodes are ordered. Immovable nodes are displayed with shadows (b). One more constraint is added so that the two topmost nodes are aligned. After these modifications, the final layout (c) is computed.

In both of the examples shown above, the same population (= 200,) crossover rate (= 0.8,) and mutation rate (= 0.006) were used.

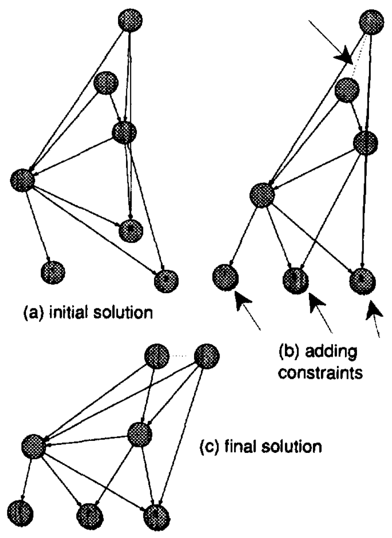


Figure 6: Manual positioning of nodes

5 Discussions

5.1 Advantages of using GALAPAGOS

Just as we have shown in the previous section, GALAPAGOS can not only lay out a complicated directed graph automatically, but also allow users to edit the result and modify the constraints to get preferred results. This is a very promising approach toward computer-aided graphic layout system. Here are the advantages of using GALAPAGOS for graphic object layout problems.

- No need for special ad-hoc layout algorithms

  You don't have to devise a complicated algorithm to lay out objects. The only thing you have to specify is an evaluation function which corresponds to the criteria you choose. Even when the criteria should be changed, you don't have to change the algorithm at all, and all you have to modify is the evaluation function.

- Strength of constraints

  If you have several criteria which are not of equal importance, you can specify the importance of each criterion by assigning a different value to each criterion. For example, when laying out tree nodes, node overlapping is considered to be worse than an unbalanced node. To represent the difference of importance, you can add a big value to the return value of the evaluation function when there are overlapping nodes, while you add a small value when you find an unbalanced subtree.

- Robustness

  Genetic algorithms always give an answer which they think is closest to the solution at the moment. They give an answer even when there's no optimal solution at all to the constraints given to them. Although you cannot always expect the best solution, you have good chance of getting a good solution from the algorithm.

- Global layout considerations

  Most of the layout algorithms use only local information to determine where to lay out an object, because global considerations will lead to a combinatorial explosion in computation time. On the other hand, genetic algorithms can find a globally good solution from multiple genes which represent locally good solutions, without using excessive computation time. For this reason, they can be used for inherently tough, NP-hard layout problems.

- Interactive changes of constraints and layouts

  Constraints can be modified interactively during the calculation of graphic layout. That is, if you don't like the results computed from the algorithms, you can add or modify the constraints so that other results are produced. You can also lay out some parts of the layout fully by manual, without letting the system lay out automatically. This means that you can choose any point between fully automatic and fully manual layout.

The first four advantages came from the power of GA, and the last one came from GA's interactive extension introduced in GALAPAGOS.

5.2 Using different GA

Although GALAPAGOS is using the same algorithm as the GENESIS system, the modification is minimal and any other GA algorithm can be used for GALAPAGOS. If an advanced GA algorithm which shows good performance in graphic layout problems is found, it can easily be incorporated in the new version of GALAPAGOS.

5.3 Selection of the evaluation function

The evaluation function changes the behavior of genetic algorithms dramatically. Sometimes you cannot get a expected result because of the improper evaluation function. This may seem to be a problem, but it can also be considered to be a strong point of the algorithm because of the following reasons.

- Users can sometimes get unexpected but desirable layout.
- Users can find the relation between the constraints and the result layout. From the relation, users can analyze what constraints make the layout look good, from another point of view.

6 Related works and future works

Genetic algorithms are becoming popular in the area of VLSI layout systems [2] [14]. Some VLSI layout systems are using the combination of genetic algorithms and simulated annealing [7] [9]. Using genetic algorithms for the layout of a directed graph is reported in [5] [13], but the layout computed by those systems are not very good. Kosak et al.[10] used genetic algorithms running on The Connection Machine for the layout of a network-diagram, a variant of a directed graph. They made some modifications to the standard genetic algorithms, just like the ones shown in this paper, to perform efficient computation. They used "perceptual organization" for the layout of a network-diagram, which is a structure which gives users better understandings about the network.

Genetic algorithm work well for all those systems shown above as an automatic layout algorithm. However, these systems shown above do not allow users to modify the evaluation function at run time, and users cannot tell their preference to the system during computation.

Takahashi et al.[15] created an automatic graphic layout system called TRIP2, where users can edit the output structure and tell the system that the structure is modified. TRIP2 is novel because not only a data structure is nicely visualized by the system, but also user feedback to the output can modify the original data structure, and bidirectional communication between the user and the layout system is possible. However, the automatic layout system used in TRIP2 is based on a rather simple constraint solver, and complicated constraints like those shown in this paper cannot be solved. Moreover, the layout procedure is fully automatic and users cannot tell the system their preferences during the layout procedure.

Our future work includes putting together all those nice features proposed in this paper and others, and construct a general-purpose graphic layout system using genetic algorithms. In the current GALAPAGOS system, users can specify only three constraints in addition to the default constraints. We are planning to use a simple interpreter language to allow users to write any kind of arithmetic expressions for the specification of constraints.

7 Conclusions

Genetic algorithms are simple, robust and powerful algorithms for laying out graphic objects. Although genetic algorithms are not very fast and not always reliable, they can be used in many graphic layout applications where deterministic layout algorithms are not easily created. They can also allow the users to modify the evaluation function at run time to change the constraints for the layout interactively. We have developed a graphic layout system GALAPAGOS and proved the usefulness of genetic algorithms for graphic layout problems.

8 Acknowledgements

We thank Dr. Hiroaki Kitano of NEC for introducing us the power of genetic algorithms. We also thank Dr. Toru Chiba and Dr. Toru Kawata of SHARP Corporation for the support of our research.

References

[1] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Hillsdale, NJ, July 1987. Lawrence Erlbaum Associates, Publishers.

[2] J. P. Cohoon and W. D. Paris. Genetic placement. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 422–425, 1986.

[3] David E. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[4] John J. Grefenstette and Nicol N. Schraudolph. *A User's Guide to GENESIS 1.1ucsd*. Computer Science & Engineering Department, University of California, San Diego, La Jolla, CA, August 1990.

[5] L. J. Groves, Z. Michalewicz, P. V. Elia, and C. Z. Janikow. Genetic algorithms for drawing directed graphs. In Z. W. Ras M. Zemankova M. L. Emrich, editor, *Methodologies for Intelligent Systems 5, Proceedings of the Fifth International Symposium*, pages 268–276. North-Holland, October 1990.

[6] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[7] Youngtak Kim, Youngjo Jang, and Myunghwan Kim. Stepwise-overlapped parallel annealing and its application to floorplan designs. *Computer Aided Design*, 23(2):133–44, March 1991.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, (220):671–680, 1983.

[9] S. Koakutsu, Y. Sugai, and H. Hirata. Block placement by improved simulated annealing based on genetic algorithm. *Transactions of the Institute of Electronics*, J73A(1):87–94, January 1990.

[10] Corey Kosak, Joe Marks, and Stuart Shieber. A parallel genetic algorithm for network-diagram layout. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 458–465, UCSD, California, July 1991. Morgan Kaufmann Publishers.

[11] Brad A. Myers, Brad Vander Zanden, and Roger B. Dannenberg. Creating graphical interactive application objects by demonstration. In *UIST89*, pages 95–104, November 1989.

[12] Dan R. Olsen and Kirk Allan. Creating interactive techniques by symbolically solving geometric constraints. In *UIST90*, pages 102–107, October 1990.

[13] Donald P. Pazel. A graphical interface for evaluating a genetic algorithm for graph layout. Technical Report RC14348, IBM Research Division, T.J. Watson Research Center, 1989.

[14] Khushro Shahookar and Pinaki Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transaction on Computer-Aided Design*, 9(5):500–511, May 1990.

[15] Shin Takahashi, Satoshi Matsuoka, Akinori Yonezawa, and Tomihisa Kamada. A general framework for bi-directional translation between abstract and pictorial data. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 165–174, October 1991.

[16] Roberto Tamassia and Peter Eades. Algorithms for drawing graphs : an annotated bibliography. Technical Report CS-89-09, Brown University, Department of Computer Science, October 1989.

[17] Bradley T. Vander Zanden. Constraint grammars - a new model for specifying graphical application. In *Conference on Human Factors in Computing Systems (CHI 89)*, pages 325–330. ACM SIGCHI, Addison-Wesley, May 1989.