Integration of Declarative Approaches (System Demonstration)

Arne Frick[1] *, Can Keskin, Volker Vogelmann[2]

[1] Tom Sawyer Software, 804 Hearst Avenue, Berkeley CA 94710, EMail: africk@tomsawyer.com
[2] Universität Karlsruhe, Institut für Telematik, Postfach 6980, D-76128 Karlsruhe, Germany EMail: vogelmann@teco.uni-karlsruhe.de

Abstract. This demonstration shows the GOLD system, an extensible software architecture integrating several declarative layout strategies, including spring-embedders, local constraints and genetic algorithms. The underlying paradigm is to consider graph layout problems as geometric constraint satisfaction problems [4].
In addition to satisfying global aesthetics criteria, the system allows for the interactive specification of local criteria per vertex (edge).

1 Introduction

In general, spring embedder algorithms produce "good" drawings of undirected graphs according to the following global criteria

- edge lengths should be approximately equal,
- adjacent vertices should be closer together than non-adjacent ones, and
- inherent symmetries should be displayed.

In many cases, however, there exist aesthetically more pleasing drawings, which may rely on other aesthetic criteria unavailable to spring embedder algorithms. Often, such criteria can be expressed declaratively. Declarative graph drawing strategies include *simulated annealing* [3], *constraint-satisfaction* [10], *Graph Grammars* [1], and *genetic algorithms* [11].

In [7], a proposal was made towards the integration of declarative and algorithmic approaches. Their goal was to combine the strengths, while overcoming their respective difficulties. Algorithmic approaches are fast, but difficult to modify. Declarative approaches, on the other hand, are usually easy to adapt to changing user requirements, but inherently slow, as they employ general problem-solving frameworks.

2 Genetic algorithms for Graph Drawing

In this section, we briefly review genetic algorithms and show how to apply this paradigm to the field of graph drawing.

* This research was performed while the author was working at Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation.

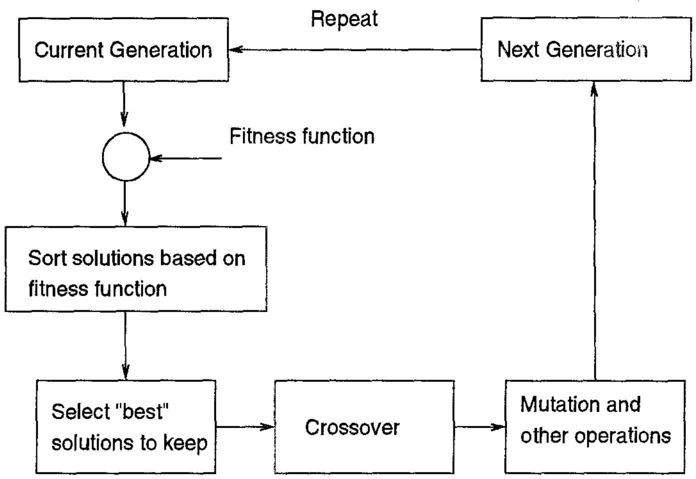
Current Generation
Repeat
Next Generation
Fitness function
Sort solutions based on fitness function
Select "best" solutions to keep
Crossover
Mutation and other operations

Fig. 1. The basic steps in a genetic algorithm.

2.1 Overview

Genetic algorithms (GAs) [5] belong to a family of optimization techniques known as *evolutionary algorithms (EAs)* that are based on principles of natural evolution. Maintaining a *population* of potential solutions, they perform a *selection* operation based on the *fitness* of individuals, modified by *recombination* and *mutation* operators to respect genetic diversity.

A GA can be interpreted as a modified random search. As with *simulated annealing (SA)*, it is a probabilistic method to solve hard optimization problems. These algorithms do not guarantee the optimum value, but the error probability can be made arbitrarily small.

The application of GA's to optimization problems requires a suitable encoding of candidate solutions. Traditional GA's represent possible solutions by binary bit strings (*chromosomes*), while newer work is also investigating other forms of representation as well. The initial population for a GA search is usually selected at random.

The basic structure of a GA is shown in 1. A part of the population is selected based on the principle of *survival of the fittest* by an *objective* or *fitness* function. The population size affects both the performance and the efficiency of the GA. A large population discourages premature convergence to suboptimal solutions. On the other hand, a large population requires more evaluations per generation, resulting in slower convergence. Crossover rate and mutation rate also affect the convergence to suboptimal solutions. The *generation gap* controls the percentage of the population being replaced during each generation according to a *selection strategy*.

GAs are known to be slow in practice, which is not really a surprise, since they are used to solve hard optimization problems. In many cases, the efficiency of a GA can be enhanced by combining it with other heuristics. For example, the initial population can be seeded with the results of a problem-specific heuristic.

2.2 Constraint satisfaction using genetic algorithms

Despite their common problems achieving satisfactory runtime efficiency, genetic algorithms have traditionally been used for constraint solving problems in cases where optimal or near-optimal solutions are required.

Genetic algorithms being used for constraint-solving problems have to deal with the problem of candidate solutions violating the constraints [12]. The most popular strategy is to generate potential solutions without considering the constraints at first. In a second phase, candidate solutions in violation of constraints are penalized by reducing their fitness. Other strategies have drawbacks exist, but have drawbacks such as being computationally expensive or posing problems to express the constraints.

2.3 Graph Drawing with genetic algorithms

This section shows how to use genetic algorithms to solve graph drawing problems. Usually, the formulation of a graph drawing problem as a GA involves the solution of a numeric optimization problem.

Previous research in this area includes [6, 8, 9] and [11]. In [6], a parallel GA for network-diagram layout is presented, in which perceptual organization is preferred over aesthetic layout. For a directed graph with 12 nodes, it took about 2 minutes to create a 2-D layout on 4096-processor machine. In [9] a GA is used for interactive two dimensional directed graph layout. The user can modify constraints like "two specified nodes have the same x-coordinate". Although the parameters (i.e. crossover rate, and mutation rate) were published, no data on the computational complexity of their approach is available. In view of the remarks make above regarding the tendency of GA's to be computationally slow, it is probably safe to assume that the runtime complexity does not stand out in particular.

To apply the GA paradigm to graph drawing, a set of candidate graphs is maintained as the population. Global aesthetics and local criteria are expressed as geometric constraints. In order to apply the penalty method these constraints become part of the fitness function (also called "energy constraints" in [14]). Each candidate graph is then evaluated and assigned a fitness value. The problem of minimizing the fitness function is equivalent to finding a solution for the constraint set.

The use of constraints provides a mechanism for quantifying the quality of a given layout, thus allowing objective comparison of the quality of two layouts.

3 The GOLD Architecture

The aim of GOLD is to combine the high speed of the spring-embedder paradigm with the ability of other declarative approaches to consider arbitrary geometric constraints in addition to the few global criteria employed by spring-embedders. To this end, we have developed a flexible, modular architecture comprised of three components (cf. Fig. 2):

- A *graphical user interface* (GUI) module provides for control mechanisms to steer the layout process, as well as a mechanism to visually specify local constraints on the graph.

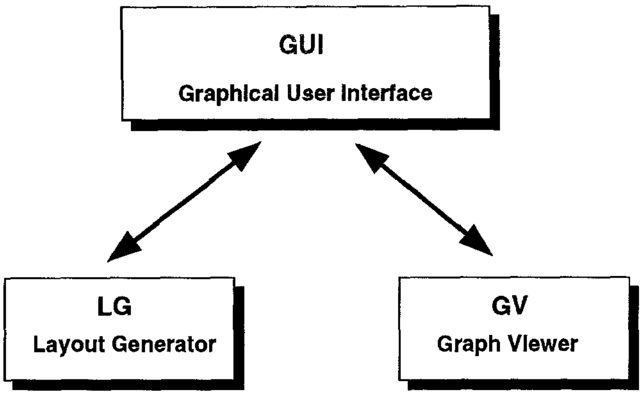

Fig. 2. The GOLD architecture.

- Several *layout generator* (LG) modules. Each LG module realizes a graph drawing strategy. In principle, arbitrary strategies conforming to a common protocol could be plugged in here. According to our goal stated above, we have implemented a random layout module to get uniform input graph layouts, a GEM-3D spring embedder module according to [2], and a genetic algorithm module. For the purpose of research an incremental constraint solver is also integrated, which solves local constraints with heuristic techniques.
- A *graph visualization* (GV) module is based on GeomView [13], an interactive 3-D geometry viewer with the usual rotation, translation and zoom mechanisms for interactive exploration.

The system creates 3-D drawings. All modules are realized as separate processes and are exchangeable. The key features of GOLD are

- extensibility. Layout strategies and viewer modules can be added or exchanged easily.
- the ability to specify constraints textually and visually. Vertices, edges and groups of these can be selected and assigned constraints.
- an extensible set of predefined local constraints.
- perspective 3-D views with real-time interactive graph exploration, e.g. navigation, zoom, camera flights.

In the GOLD system, global aesthetic constraints such as the desired inter-node distance and edge length are pre-defined. In addition, an extensible set of local constraints includes the following:

- Edges should enclose a specified angle.
- Nodes should lie in a plane.
- Nodes should lie on a line.
- Desired edge length for a particular edge.
- Equal length of a set of edges.

is provided. There also exist user controls for quality, speed, and constraint priority (i.e. aesthetic tradeoffs).

4 Applications

Layout creation in GOLD is usually performed in two distinct phases. In the first phase, only global constraints are active. Second, both global and local constraints are active. Then, local constraints are considered to have a higher priority than global constraints, as they are specified by the user, and we would like to take these constraints as a strong hint as to what the user would like to see.

This two-phase approach allows factoring out the first phase and use a fast spring-embedder algorithm. Compared to the use of genetic algorithms or spring-embedders by themselves, the integration of both combines their respective strengths, i.e. it leads to better results than spring embedders, while being faster than traditional genetic algorithms. Fig. 3 demonstrates this using two examples. For example, Fig. 3a shows a 3-D drawing of a wheel, which a user would like to see as a planar drawing. Using only a 3-D spring embedder, it is virtually impossible to achieve a plane drawing. Adding a few simple constraints, however, achieves the desired result.

The remainder of this section describes two experiments to measure the speedup and quality improvement achieved by the integrated strategy and the spring-embedder approach, respectively.

The first experiment measures the convergence speed and quality are measured for the graphs in Fig. 3. Each graph was drawn using two different initial seedings for the GA. The result for the Wheel is shown in Fig. 4. The figure indicates a significant speedup and quality improvement for an initial seeding based on the output of the GEM-3D spring embedder algorithm, compared to a random initial seeding. The corresponding figure for the Chair graph, which cannot be shown here for lack of space, displays the same behavior.

In the second experiment, the quality of four different graph layouts was measured both visually and by their respective fitness values. The constraints used in these examples were as follows. The Chair example (see Fig. 5) was solved by constraining its legs to have equal distance, and the nodes of its back to fall within the same plane. The Tree (see Fig. 7) was solved by defining a preferred direction for edges. The House (see Fig. 6) was solved by a set of constraints on angles, plane and lines. The hardest graph was the Wheel (see Fig. 8). To make it look like a planar wheel in 3-D, constraints were defined that restrict its spokes and the outer edges to have equal lengths, respectively.

The performance results are given in Table 1, which displays the input parameters and the resulting fitness, the latter being defined as the logarithm of the sum of all penalty terms, i.e. lower fitness values indicate better results. The data suggest that the Wheel is the hardest problem in this experiment: Although it took much longer, its fitness is worse than the other's.

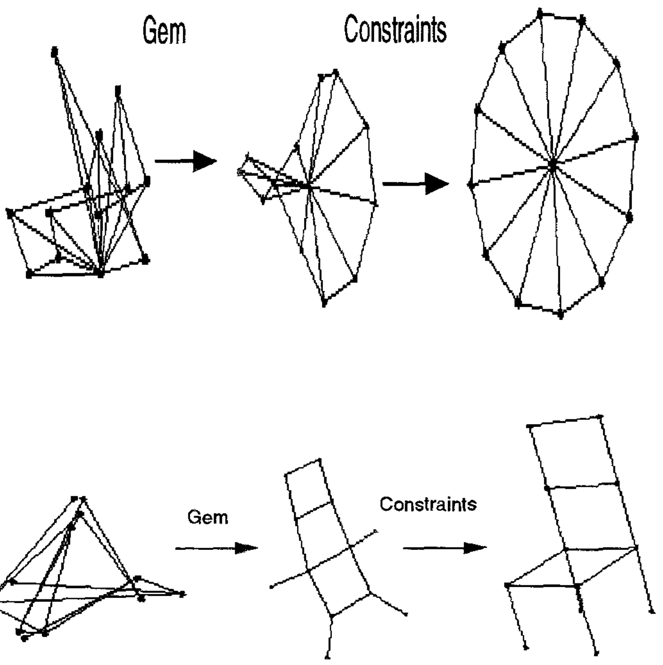

Fig. 3. Two results of the integrated GA approach with a spring-embedder used to seed the GA. The quality improvement is clear.

5 Summary

The use of spring-embedder algorithms allows to quickly generate initial drawings. Measurements indicate large speedups for the integrated approach, as compared to randomly-seeded genetic algorithms. Also, we have found evidence of quality improvements of an integrated approach vs. an approach that was based on combining spring-embedders with local constraint propagation techniques. For example, the 61-vertex Wheel in Fig. 8 was no problem at all for the integrated approach, while 13-vertex Wheel was found to be a hard instance of the combination of spring-embedders with local constraints. Overall, the GOLD architecture has proved to be a valuable tool to explore the effects of combining several layout strategies.

6 Acknowledgements

We would like to thank Frank Schwellinger for his initial implementation of our genetic algorithm in his M.Sc. thesis.

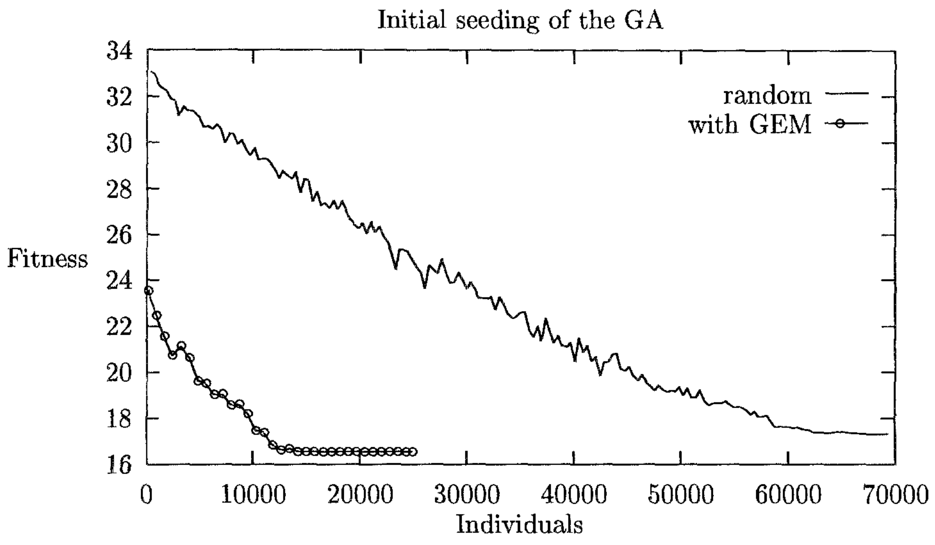

Fig. 4. Speedup and performance improvement displayed by a spring-embedder seeded GA, versus one seeded randomly. The measurement was performed on a 61-vertex Wheel.

Parameter	Chair			House		Tree		Wheel	
Population Size	15	20	50	20	50	20	20	5	5
Mutation Rate	0.0	0.001	0.001	0.001	0.001	0.002	0.001	0.002	0.001
Crossover Rate	0.5	0.0	0.6	0.5	0.0	0.5	0.6	0.7	0.6
Generation Gap	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Selection Strategy	P	P	P	P	P	E	P	P	P
Fitness	10.822	10.637	10.588	17.477	17.437	14.606	15.563	23.285	23.356
Time[s]	1	2	5	1	3	3	2	58	55

Table 1. Performance of the integrated layout method based on genetic algorithms on the example graphs.

References

1. F. J. Brandenburg. Designing graph drawings by layout graph grammars. In Roberto Tamassia and Ioannis Tollis, editors, *Proceedings of Graph Drawing'94*, volume 894 of *Lecture Notes in Computer Science*, pages 416–427. DIMACS Workshop on Graph Drawing, Springer Verlag, 1995.
2. I. Bruß and A. Frick. Fast interactive 3-D graph visualization. In Franz Brandenburg, editor, *Proceedings of Graph Drawing'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 99–110. Springer Verlag, 1996.

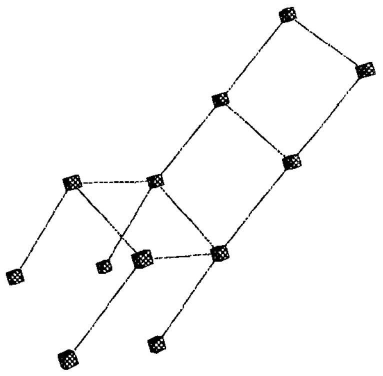

Fig. 5. Chair

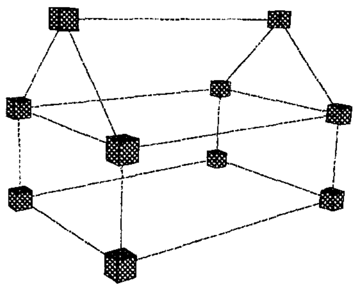

Fig. 6. House

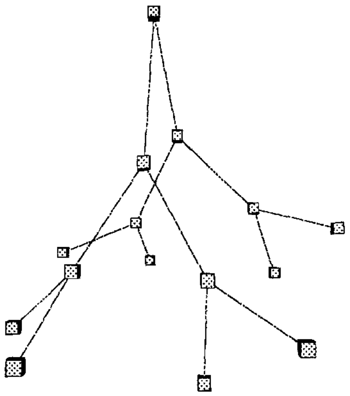

Fig. 7. Tree

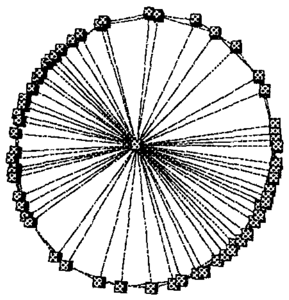

Fig. 8. Wheel

3. I. F. Cruz and J. P. Twarog. 3-D graph drawing with simulated annealing. In F. J. Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 162–165. Springer-Verlag, 1995.
4. E. Dengler, M. Friedell, and J. Marks. Constraint-driven diagram layout. In *Proceedings of the 1993 IEEE Workshop on Visual Languages*, pages 330–335, 1993.
5. J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
6. C. Kosak, J. Marks, and S. Shieber. A parallel genetic algorithm for network-diagram layout. In *Proc. 4th Int. Conf. on Genetic Algorithms (ICGA91)*, 1991.

7. T. Lin and P. Eades. Integration of declarative and algorithmic approaches for layout creation. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing*, volume 894 of *Lecture Notes in Computer Science*, pages 376–387. DIMACS, Springer-Verlag, October 1994. ISBN 3-540-58950-3.
8. E. Mäkinen and M. Sieranta. Genetic algorithms for drawing bipartite graphs. Technical report, Department of Computer Science, University of Tampere, 1994.
9. T. Masui. Graphic object layout with interactive genetic algorithms. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 74–87, Seattle, Washington, 1992.
10. S. Matsuoka, S. Takahashi, T. Kamada, and A. Yonezawa. A general framework for bidirectional translation between abstract and pictorial data. *TOIS*, 10(4):408–437, 1992.
11. Z. Michalewicz. *Genetic algorithms + data structures = evolution programs.* Springer-Verlag, 1992.
12. Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In R. Belew and L. Booker, editors, *Genetic Algorithms*, pages 151–157, 1991.
13. Phillips, Levy, and Munzner. Geomview: An interactive geometry viewer. *Notices of the American Mathematical Society*, 40, 1993.
14. A. Witkin, K. Fleischer, and A. Barr. Energy constraints on parameterized models. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 225–232, July 1987.