Combination and Integration of Declarative Approaches (Full Paper and System Demonstration)

Arne Frick[*] Can Keskin[†] Volker Vogelmann[‡]

June 3, 1996

Abstract

This paper explores techniques to improve the quality of drawings produced by spring embedder algorithms for undirected graphs. Although producing quite acceptable results in general, the spring embedder model only optimizes global aesthetics criteria.

The main idea is to combine global and local criteria to produce better results. First, we consider a two-stage approach that uses a spring embedder algorithm to initially produce a reasonable drawing, which may be subsequently improved by interactively specifying and solving graphical constraints. Experiments conducted with this approach shows that both phases may interfere with each other when executed sequentially. An obvious solution to overcome this difficulty is to integrate both phases. This is similar to the proposed integration of declarative and algorithmic approaches in [17]. Here, we use the genetic algorithm paradigm for this purpose. Unlike the similar technique of simulated annealing, our approach is reasonable fast. To our knowledge, this is the first declarative approach based on the genetic algorithm paradigm that aims at arbitrary undirected graphs.

The methods described in this paper have been implemented and explored using a flexible, extensible object-oriented software architecture.

1 Overview

The paper is organized as follows. Sect. 2 reviews three declarative methods for graph layout creation: spring embedders, constraint-solving techniques and the genetic algorithm paradigm. The combination of these techniques by satisfying global and local constraints separately is explored in Sect. 3. As an evaluation suggests, the sequential combination yields better overall results, but should be further improved. This leads to an integrated approach unifying the optimization of global and local constraints, which is described in Sect. 4. A summary concludes the paper. The appendix describes the GOLD architecture and its implementation GOLDSPEC, an interactive graphical environment to explore and assess the techniques described in the main sections.

2 Fundamentals

The problem of drawing a graph nicely can be considered as a problem of constructing a layout of nodes and arcs according to a set of aesthetics criteria.

[*]EMail: frick@ipd.info.uni-karlsruhe.de
[†]EMail: keskin@teco.uni-karlsruhe.de
[‡]EMail: volker@teco.uni-karlsruhe.de

2.1 DEFINITION
A *drawing* or *layout* of a graph $G = (V, E)$ consists of two mappings ϕ and ρ, where

$$\phi(v) : v \mapsto \mathbb{R}^n, \qquad n \in \{2, 3\}$$

maps every vertex $v \in V$ onto a position in the drawing plane (or volume, for that purpose). Also, every edge $e = (u, v) \in E$ with $u, v \in V$ is mapped onto a parameterized continuous curve

$$\rho_e(t) : t \mapsto \mathbb{R}^n, \qquad n \in \{2, 3\}, \quad t \in [0, 1]$$

with $\rho_e(0) = \phi(u), \rho_e(1) = \phi(v)$. The mapping function $\rho = \bigcup_{e \in E} \rho_e$ defines a *routing* of E.

Among commonly used global aesthetics criteria for drawing arbitrary undirected graphs are the following.

- distribute vertices evenly in the drawing area
- have approximately equal edge lengths
- have adjacent vertices be closer to a vertex than non-adjacent ones
- maximize the display of symmetries
- minimize the number of edge crossings

All of these criteria are hard to satisfy optimally, since the underlying optimization problems are $\mathcal{NP}$-hard to solve. In addition, there even exist pairs of criteria that may contradict each other, e.g. symmetry display vs. edge crossings. A common solution to overcome this difficulty and find solutions that simultaneously satisfy these criteria to some reasonable extent is to use heuristics.

A well-known and frequently used technique for drawing arbitrary undirected graphs is the *spring embedder* paradigm due to EADES [5]. It has been observed that although the paradigm declaratively seeks to satisfy the first three criteria, the solutions often satisfy one or both of the remaining two as well. As the spring embedder approach is well-known by now, we shall not go into more detail, but refer the interested reader to the original paper of Eades [5] and its subsequent improvements [13, 7]. In our work, we have found the GEM algorithm [6] with additional speed-up heuristics very helpful to create online interactive drawings in the GOLDSPEC environment (cf. Appendix).

Two other declarative approaches will be outlined in the following subsections, *constraint satisfaction* and *genetic algorithms* (GA). A third paradigm, *simulated annealing* (SA), will not be considered in detail for several reasons. The first one is lack of space. Next, SA has considerable efficiency problems and cannot be considered a candidate for online interactive applications as it stands. Last, SA shares several key properties with genetic algorithms, as we shall see below.

2.1 Constraints

Constraint satisfaction is a declarative method of specifying requirements that the objects of a domain must satisfy. In particular, a *constraint* describes a relation between the objects. For graphs, the problem domain consists of drawings of a graph G. The main challenge is to find a good formulation for aesthetics criteria as constraints. One idea to do so is to consider the graph layout problem as a *geometric constraint* problem (cf. Fig. 1). For instance, a constraint

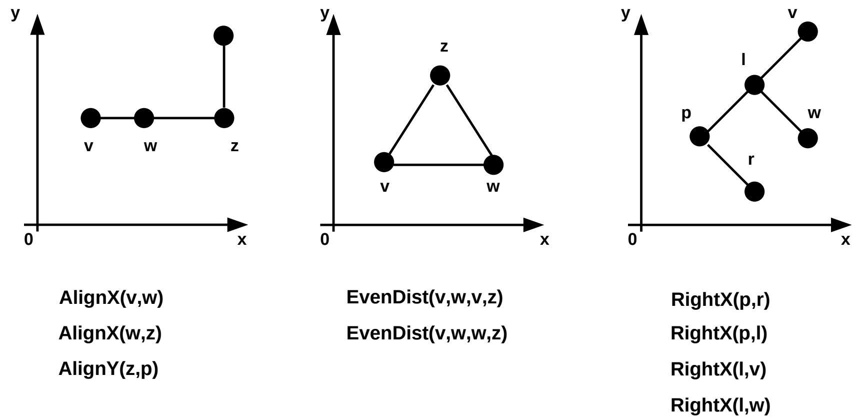

Figure 1: Examples of geometric constraints.

AlignX(v,w) on two vertices v and w requires that the coordinates of two vertices v and w should have the same x positions.

A simple constraint-satisfaction technique is the *local propagation of known states* [16], or shorter *local propagation*. This mechanism is best illustrated using so-called constraint graphs (cf. Fig. 2). The square nodes represent variables, and the round nodes represent operators. The arguments to an operator are attached on its left side, and the result is attached on its right side. Operators that take no argument are constants.

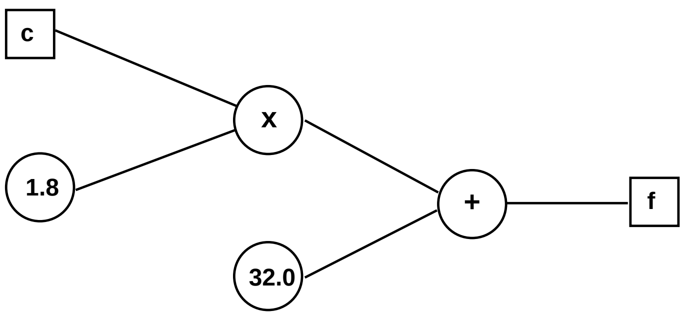

Figure 2: Example of a constraint graph for the numerical constraint $c * 1.8 + 32 = f$.

In order to solve the constraints in such a constraint graph, known values for variables are propagated along the arcs. When a node receives sufficient information from its arcs, it *fires*, calculates one or more values for the arcs that do not contain values, and sends these new values out. These new values then propagate along the arcs, causing new nodes to fire, and so on (cf. Figure 3).

Another constraint-satisfaction technique is *relaxation* [16], an iterative numerical approximation technique. Relaxation makes an initial guess at the values of unknown objects, and then estimates the error that would be caused by assigning these values to the objects. New guesses are then made, and new error estimates calculated, and this process repeats until the error is minimized.

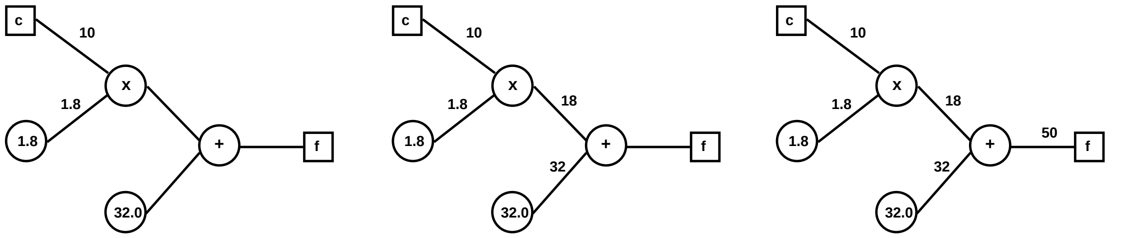

Figure 3: Constraint propagation (see Fig. 2).

In general, geometric constraints are a type of numerical constraints and can be written as equations.[1] Geometric constraint-solving is then reduced to the solution of an equation system. The resulting equations may become quite large and may not even be linear. The approach was extensively studied in [9].

2.2 Genetic Algorithms

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

2.2.1 Overview

Genetic Algorithms (GA) [10] belong to a optimization techniques known as *evolution strategies* imitating the principles of natural evolution for parameter optimization problems [25]. Evolution strategies maintain a *population* of potential solutions and base the selection of candidate solutions on a from the population have some selection process based on the *fitness* of individuals, and some *recombination* operators.

In the GA paradigm (cf. Fig. 4), a population of candidate solutions (*chromosomes*), a part of the new populations is selected based on the *survival of the fittest* principle, as determined by an objective of *fitness* function. In order to preserve or enable genetic pluralism, *mutation* and *crossover* occur with certain probability.

From an Operations Research perspective, GA's can be interpreted as a modified random search. Like *simulated annealing (SA)*, it is a probabilistic algorithm for hard optimization problems. These algorithms do not guarantee the optimum value, but the error probability can can be made arbitrarily small. To solve an optimization problem using a GA, candidate solutions must be encoded in a suitable form. Traditional GA approaches represent candidate solutions by binary bit strings. Newer work also uses other representations for efficiency reasons. The initial chromosome population for a GA search is usually selected at random.

GA's are known to be slow in practice, which is not really a surprise, since they are used to solve hard optimization problems. Naturally, many modifications and heuristics to improve their performance have been proposed. An obvious approach is to use parallelism [8, 15], since the paradigm is parallel by nature. The runtime behavior of the algorithm is influenced by its parameters. The fitness function itself plays a crucial role in the calculation and does not only

[1]The case of inequations can be reduced to this case and the application of interval arithmetic [11, 12].

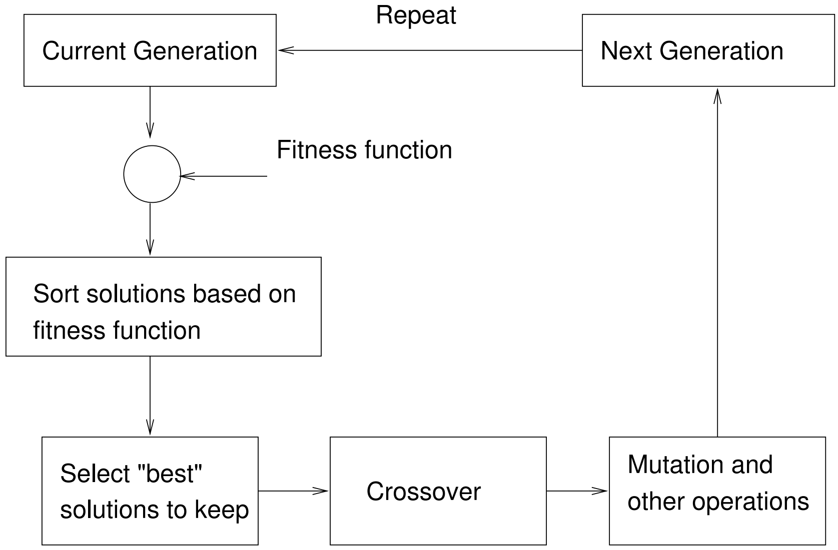

Figure 4: Architecture of a Genetic Algorithm.

influence the quality of the results, but also the overall performance of the algorithm to a great extent. We shall come back to the topic of performance improvement below.

2.2.2 Handling Constraints in Genetic Algorithms

Three different approaches to the constraint-solving problem in GAs have been proposed in [20]. One way of dealing with candidate solutions violating the constraints is to generate potential solutions without considering the constraints and then penalizing them by decreasing the fitness of the fitness function.

Another approach focuses on the use of special representation mappings (*decoders*) which guarantee that only feasible solutions are generated, and on the application of special *repair algorithms* to correct any infeasible solutions. This approach has several drawbacks. Decoders are frequently computationally intensive. Not all constraints can be easily implemented this way, and the resulting algorithm must be tailored to the particular application. The same is true for repair algorithms. We therefore stick to the first approach.

2.2.3 Hybridization

In many cases, the effectiveness of a GA can be enhanced by hybridizing it with another heuristic. Hybridization can be carried out in various ways. The idea of seeding a population with the results of applying a problem-specific heuristic can be viewed as a minimal-level hybridization. Local neighborhood search and extensions such as simulated annealing or threshold accepting (TA) [4] can be formulated as a GA in order to enhance performance. The schematic form of the TA method is as follows:

One approach is to use a GA to find regions of the solution space which are good candidates for locating the optimum [24]. Another approach is to converge to the nearest local optimum and then use the GA for eluding local optima.

Program 2.1 Algorithm implementing the Threshold Accepting strategy.

```
1  choose an initial configuration x and an initial threshold T;
2  choose a new configuration x' from the neighborhood of x;
3  if f(x') ≥ f(x) − T then set x ← x';
4  decrease the threshold T;
5  if the termination rule is satisfied, stop; otherwise go back to step 2.
```

2.2.4 Interactive Graph Drawing with Genetic Algorithms

This section shows how to use genetic algorithms to solve graph drawing problems. Previous research in this area includes [15, 18, 19] and [21]. [15] presents a parallel GA for network-diagram layout, where perceptual organization is preferred over aesthetic layout. For a directed graphs with 12 nodes, it took about 2 minutes to create a 2-D layout on 4096-processor machine. In [18], a GA is used for interactive two dimensional directed graph layout. The user can modify constraints like "two specified nodes should have the same x coordinate". They reveal their parameter choices on population, crossover rate and mutation rate, but do not comment on computation time.

We now consider the problem of applying the GA paradigm to graph drawing. To this end, aesthetics criteria are expressed as geometric constraints, which in turn become part of the penalty function

$$f_A(x) = f(x) + M^k w^T c_v(x),$$

where w is a vector of nonnegative weights, the vector c_v quantifies the magnitudes of any constraint violations, M is the number of the current generation and k is a suitable exponent. The dependence of the penalty on generation number biases the search increasingly heavily towards feasible space as it progresses.

Our interactive programming tool GOLDSPEC described in the Appendix enables the user to set various operational parameters, so the tradeoff between quality and speed is under the user's control.

3 Combination of global and local constraints

This section describes an approach to satisfy global and local constraints independently. The goal is to compute a layout that satisfies both global aesthetics criteria and local, user-specified constraints.

The main idea in this section is to use a two-stage approach to compute 3-D embeddings of undirected graphs. In the first stage, fast spring embedder algorithm as described in [2] is used to produce a globally pleasing drawing, which is then fine-tuned in the second phase by solving local constraints on the resulting drawing. The remainder of this section focuses on the second phase and show how to solve a user-specified set of constraints. The requirement here is to satisfy each single constraint. To this end, a single constraint is selected at random, and it is tried to satisfy. This process is repeated, until either all constraints are satisfied, or a given threshold on the number of iterations is hit. In each step, an increased satisfaction of a constraint is interpreted as a command to move the associated vertices.

We now give a few examples to demonstrate the use of local constraints to improve drawings. Fig. 5a shows initial layout of a simple graph. This layout should now additionally satisfy the constraint AlignX(v,w). Obviously, the degree of satisfaction of AlignX(v,w) can be increased by moving the associated vertices v and w along the arrow directions as indicated in Fig. 5b. The result is shown in Fig. 5c. Analogous considerations show the possible solution of constraint EqualDistance(p,q,r,s) (cf. Fig. 5d-f). An extension of this heuristic procedure from single constraints to a set of constraints can be defined by randomly selecting a constraint from the set, which is subsequently improved. Fig. 6-7 show an example for the set C={AlignX(v,w), AlignX(w,z), EvenDist(v,w,w,z)}. The selection and improvement steps are iterated as necessary.

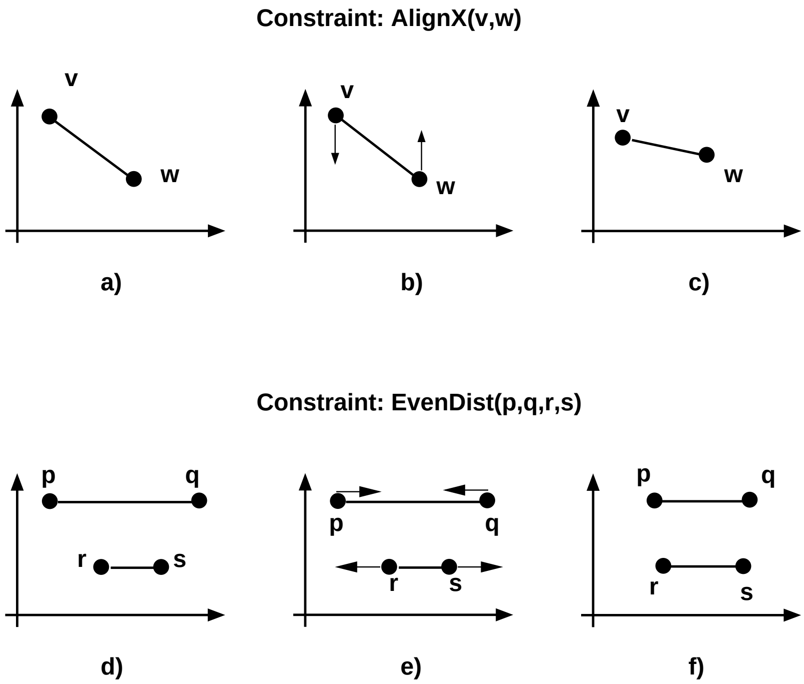

Figure 5: Two examples of constraint satisfaction. The upper half shows a single improvement step on constraint AlignX(v,w), and the lower half an improvement on EqualDistance(p,q,r,s).

Figures 8-14 show results achieved by this two-phase sequential approach, contrasted to the results achieved by using GEM only.

The separation of GEM and the local constraint propagation technique splits the satisfaction of global and local constraints. As Fig.15 shows, this may not lead to a satisfactory overall layout. In general, the idea of satisfying one constraint at a time and independent of existing global constraints may disturb other constraints or even the global layout (cf. Fig. 6). The next section resolves these problems by integrating the solution of both global and local constraints.

4 Integration of global and local constraints

This section describes a model integrating both global and local constraints. The model is shown to work by implementing it as a Genetic Algorithm.

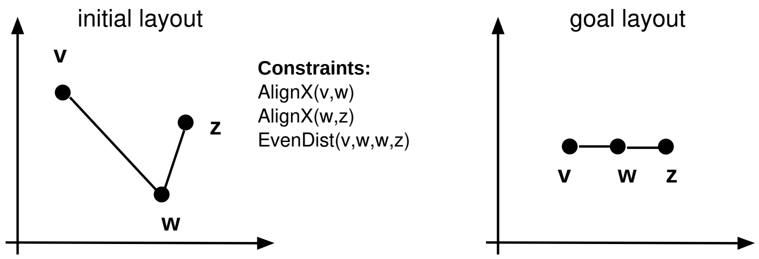

Figure 6: Initial and desired layout for a given set of constraints.

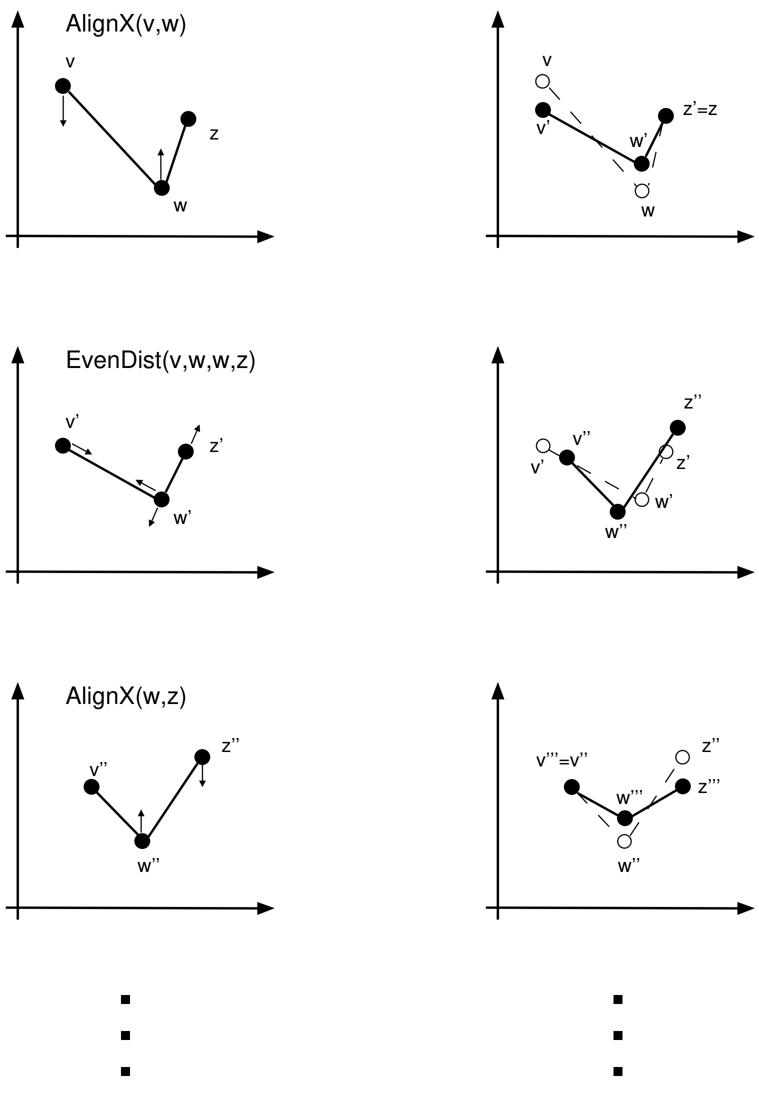

Figure 7: This figure shows an iteration to improve the constraint set C={AlignX(v,w), AlignX(w,z), EvenDist(v,w,w,z)}. The picture should be read as follows: Constraints are selected from top to bottom. The right-hand side respectively shows the state before and after improvement.

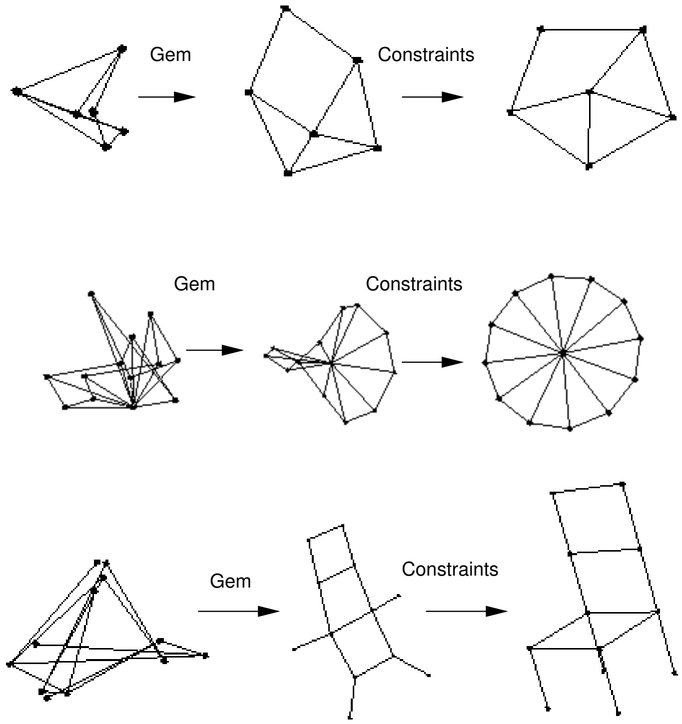

Figure 8: Results achieved with the two-phase approach, in contrast to drawings generated by a spring embedder only.

4.1 Fitness calculation

In this approach, global and local constraints are solved simultaneously by transforming potential solutions into a form suitable for the GA and using penalty functions. Assigned weights to the constraints play the role of penalties if a potential solution does not satisfy them.

4.1.1 Global Constraints

As mentioned before, an application of the GA paradigm depends on the ability to incorporate the desired global aesthetics criteria into the fitness function. From the list of criteria given in Sect. 2, we select the following subset suitable for creating expressive 3D layouts, as been shown for the spring embedder paradigm elsewhere [2].

1. The nodes should be even distributed.

2. Edges should have even lengths.

3. The distance between non-adjacent nodes 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 fitness function.

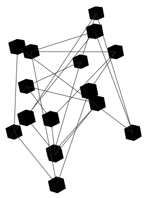

Figure 9: GEM layout of a grid-like structure.

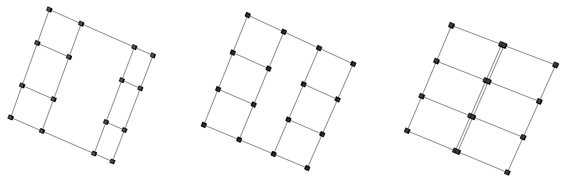

Figure 10: Three different constraint-based layouts for the graph in Fig. 9 with increasing number of constraints.

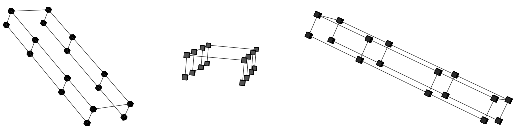

Figure 11: Another example of a constraint-based layout for the grid-like structure from Fig. 9. This time, one layout is shown from different angles.

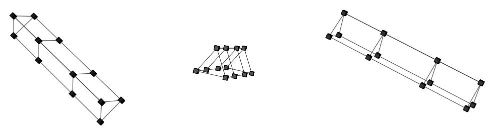

Figure 12: Results achieved with the two-phase approach, in contrast to drawings generated by a spring embedder only.

Figure 13: GEM layout of a disconnected graph.

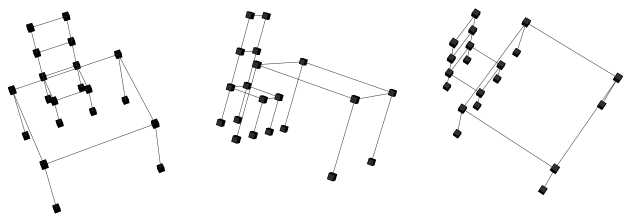

Figure 14: Successful application of the two-phase approach to the graph in Fig. 13.

The first constraint can be realized by demanding the minimum distance between nodes to be a constant value

$$l_{magic} = \min_{i \neq j}\{d(v_i, v_j)\}.$$

The goal is to equally distribute the vertices in a volume of width a, height b and depth c. To this end, we fill the space with equal-sized tetrahedrons of width

$$\sqrt[3]{\frac{12abc}{\sqrt{2}n}},$$

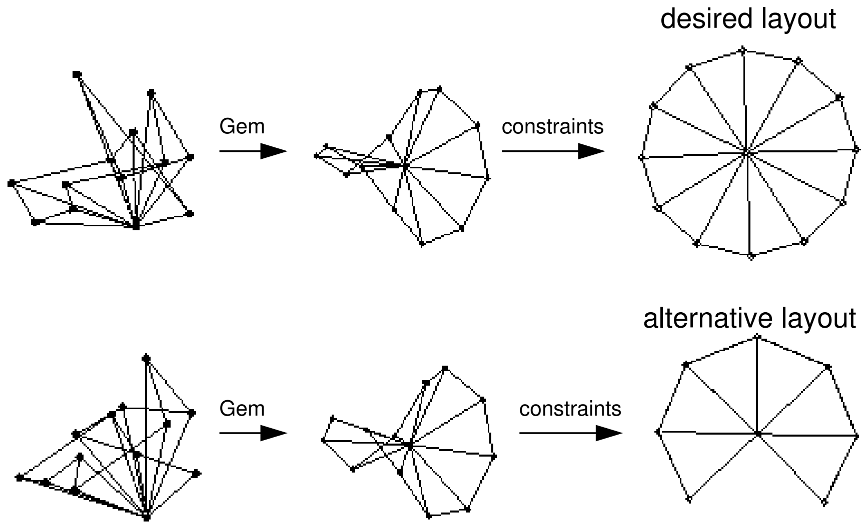

Figure 15: Negative example. The two-phase approach can fail to produce the desired layout by placing vertices on top of each other. The reason is the separation of global and local constraints.

which we call l_{magic}. Observing that the number of tetrahedrons is approximately equal to the number of crossing points, placement of vertices on such a crossing point results in an equal distribution of vertices, thus satisfying our first criterion. The remaining two criteria are met if the desired distances of nodes in dependence of their adjacencies can be chosen to be

$$\forall i,j : \begin{cases} (v_i, v_j) \in E & \Rightarrow \quad d(v_i, v_j)) = l_{magic} \\ (v_i, v_j) \notin E & \Rightarrow \quad d(v_i, v_j)) \geq kl_{magic} \text{ with } k \gg 1 \end{cases}$$

4.1.2 Local Constraints

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

1. Edges should enclose a specified angle.
2. Nodes should lie in a plane.
3. Nodes should lie on a line.
4. Desired edge length.
5. Even length of a set of edges.

4.2 Implementation of the GA

Several improvements are proposed for enhancing the effectiveness of a GA [15]. The following strategies are investigated for improving the performance:

- Refinement of the parameters.

- The use of a gradient-descent method to achieve convergence to locally optimal points in the search space.

- The initial seeding of the population.

In order to seed the population with our fast spring embedder as a problem-specific heuristic the fitness function is changed. The global constraints are adapted to *accept* the result of GEM as a *global optimum*. This is achieved by focusing on the following two aesthetic criteria:

1. Edges should have even lengths.

2. Nodes should not get too close to each other.

We have already shown how to satisfy the first criterion in general. To accept the GEM output, l_{magic} is now defined as the average length of edges created by GEM. The second component is the sum over all pairs of nodes of a function that is inverse-proportional to the distance between nodes (cf. [3]).

This procedure has the following advantages:

- The user has an initial structured graph layout in short time for contributing local constraints.

- It is expected that local constraints now are easier to handle for the GA.

Our approach is starting with a population of candidate graphs which are generated by random movements of a random set of nodes of the initial solution (GEM). Then we relax these candidates to the nearest local minimum with TA. Using the relaxed fitnesses, a fraction of the population is selected as "parents". The next generation of candidate graphs is produced by "mating" these parents. The process is repeated until the ground state graph is located.

4.2.1 Genotypes for Graphs

The choice of mating procedure is essential to the construction of a genetic algorithm. An efficient algorithm should preserve important properties of the parent clusters to the children. A common choice is to first map the structure onto a binary number string, then use string recombination as a mating procedure. This is not very efficient, however, when used to optimize the geometry of graphs. This is because the mating operation does not preserve the characteristics of the parents. In the present work, we represent a graph layout by an index-ordered list of its constituent vertices.

4.2.2 Recombination

Let D and D' be two drawings of graph G. The mating operator $P : P(D, D') \rightarrow D''$ combines both drawings as follows. First, a node v is chosen randomly by its index. This implies a partition of the node lists of D and D' *at the same index*. A new drawing D'' is now constructed by taking the positions of vertices with lower index from D and the positions of vertices with higher index from D'.

4.2.3 Mutation

It may be necessary to mutate the members of the population with a certain, low probability. We define a mutation operator which performs one of two functions with equal probability. The first mutation function moves a node according to a random vector, the second randomly moves a set of nodes in the search space.

4.3 Experience and results

The first experiment used a GA with random seeding that was given the task of computing layouts for about 30 different graphs. We optimized the control parameters for the GA. The population size affected both the performance and the efficiency of the GA. A large population discourages premature convergence to suboptimal solutions. But a large population required more evaluations per generation, resulting in an unacceptably slow rate of convergence. In the experiments, the population size ranged from 1 to 50. Also we experimented with crossover rates from 0.0 to 1.0. If the crossover rate was too low this leaded to convergence to suboptimal solutions. The influence of the mutation rate was not so heavy. This may be because of the TA does also a mutation, but in smaller steps. The generation gap controls the percentage of the population to be replaced during each generation. The experiments compared two selection strategies. P for a pure selection and E for an elitist strategy.

Then we adapted the GA to GEM as described below. The GA accepted the spring embedder solution as a global optimum after few iterations. Now we defined the local constraints. Interestingly, the result has been a dramatic speedup compared to random seeding (cf. Fig. 16 and 17).

We selected four graphs whose layout raised problems using the combined approach introduced in Sect. 3. The performance results are given in Table 1, which is read as follows. The fitness is given as the logarithm of the sum of all penalty terms. i.e. lower fitness values are better. The data indicate claim that the Wheel is the hardest problem in this experiment: Although it took much longer, its fitness is worse than the other's.

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.

The Chair example (cf. Fig. 18) was solved by constraining its legs to have equal distance, and the nodes of its back to fall within the same plane. The Tree (cf. Fig. 20) was solved by defining a preferred direction for edges. The House (cf. Fig. 19) was solved by a set of constraints on angles, plane and lines. The hardest graph was the Wheel (cf. Fig. 21). 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. Note that the figure shows a wheel with 61 vertices, a much larger

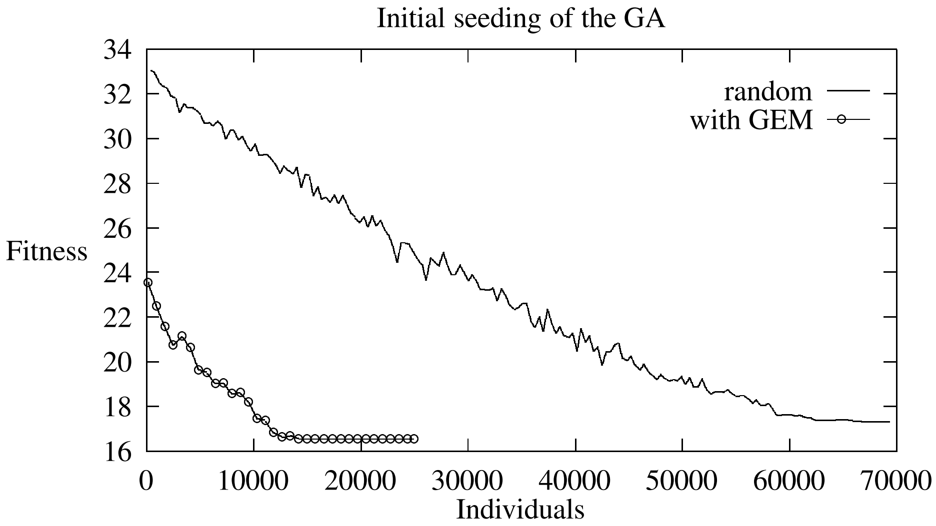

Figure 16: Speedup for Wheel (global and local constraints).

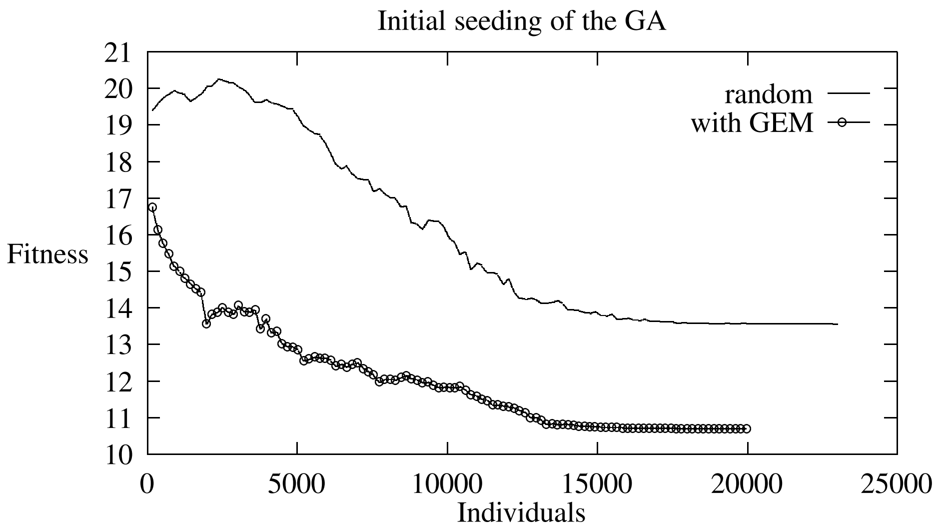

Figure 17: Speedup for Chair (global and local constraints).

instance than the 13-vertex graph considered a bad instance in Sect. 3, but no problem at all for the GA with *Gem* output as seed.

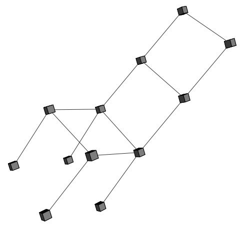

Figure 18: Chair.

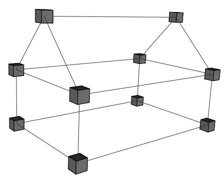

Figure 19: House.

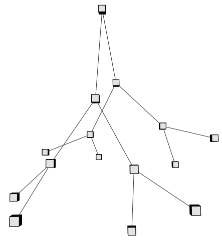

Figure 20: Tree.

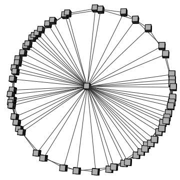

Figure 21: Wheel.

5 Related Work

There is few related work on the integration of several declarative approaches. The authors of [14] formulated constraints as additional spring-forces of a spring embedder algorithm. However, they only add angle forces to the usual vertex-repulsion forces and the oscillation heuristic of [6].

The integration of algorithmic and declarative approaches was first proposed by [17]. They introduced so-called *customizable methods* using declarative mechanisms to specify further restrictions on the output of an algorithmic method. Our method is customizable in this sense and can be easily enhanced by algorithmic approaches.

The authors of [17] noted two limitations on their approach. First, they found it to be difficult to specify constraints based on textual descriptions. Second, they remark that each customizable method can only satisfy a limited number of requirements.

Our approach overcomes both limitations. First, we have created a visual environment to specify constraints. The GOLD software architecture (cf. Appendix) is extensible and flexible in order to allow the user to choose a suitable subset of local constraints on top of algorithmic methods.

6 Summary

This paper has explored methods and techniques to integrate several declarative approaches. To this end, we combined the global constraints imposed by spring embedder algorithms with a local constraint propagation technique in a sequential way. This led to undesirable interactions between both phases. As a solution to this problem, we designed a genetic algorithm to integrate both phases. To our knowledge, this is the first successful attempt to employ genetic algorithms to the problem of drawing arbitrary undirected graphs. We feel that the GOLD software architecture (cf. Appendix) provides an excellent environment to further study the integration of algorithmic and declarative methods for layout creation.

7 Acknowledgements

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

References

[1] Franz Brandenburg, editor. *Proceedings of Graph Drawing'95*, volume 1027 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[2] Ingo Bruß and Arne Frick. Fast interactive 3-D graph visualization. In Brandenburg [1], pages 99–110.

[3] R. Davidson and David Harel. Drawing graphs nicely using simulated annealing. Technical Report CS89-13, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1989. revised July 1993, to appear in Communications of the ACM.

[4] G. Dück and T. Scheuer. Threshold accepting: A general purpose optimization appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[5] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[6] Arne K. Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs. In Tamassia and Tollis [26], pages 388–403.

[7] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software – Practic and Experience*, 21, 1991.

[8] H.W. Gellersen, C. Zeidler, W. Gerteis, B. Achauer, V. Vogelmann, and M. Staudenmaier. Objekte in verteilten systemen. Technical report, University of Karlsruhe, 1993.

[9] Allan Heydon and Greg Nelson. The juno-2 constraint-based drawing editor. DEC SRC Technical Report 131a, Digital Systems Research Center, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, December 1994.

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

[11] E. Hyvönen. Constraint reasoning based on interval arithmetic: the tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.

[12] F. Benhamou J.Older. Applying interval arithmetic to real integer and boolean constraints. *Logic Programming: The Alp Newsletter*, 6(2), 1993.

[13] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31, 1989.

[14] Thomas Kamps and Jörg Kleinz. Constraint-based spring model algorithm for graph layout. In Brandenburg [1], pages 349–360.

[15] C. Kosak and J. Marks. A parallel genetic algorithm for network-diagram layout. In *Proc. 4th Int. Conf. on Genetic Algorithms (ICGA91)*, 1991.

[16] Wm Leler. *Constraint Progamming Languages Their Specification and Generation*. Addison-Wesley Publishing Company, Inc., 1988.

[17] T. Lin and P. Eades. Integration of declarative and algorithmic approaches for layout creation. In Tamassia and Tollis [26], pages 376–387.

[18] 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.

[19] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, 1992.

[20] Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In R. Belew and L. Booker, editors, *Genetic Algorithms*, pages 151–157, 1991.

[21] Erkki Mäkinen and Mika Sieranta. Genetic algorithms for drawing bipartite graphs. Report A-1994-1, University of Tampere, 1994.

[22] John Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing, Inc., 1994.

[23] Phillips, Levy, and Munzner. Geomview: An interactive geometry viewer. *Notices of the American Mathematical Society*, 40, 1993.

[24] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, 1995.

[25] H.P. Schwefel. Evolution strategies. In *Annals of Operations Research*, volume 1, pages 165–167, 1984.

[26] Roberto Tamassia and Ioannis Tollis, editors. *Proceedings of Graph Drawing'94*, volume 894 of *Lecture Notes in Computer Science*. DIMACS Workshop on Graph Drawing, Springer Verlag, 1995.

A An interactive graphical environment to nicely draw graphs

This appendix describes the GOLD software architecture and GOLDSPEC, an interactive tool based on GOLD for visualizing graphs in 3-D. It realizes the ideas described in this paper and demonstrates that our approach is feasible for interactive use. The key features include

- extensibility. New layout strategies may be added easily using the GOLD architecture (cf. Fig. 22)
- several layout strategies, among which the user may choose at will (cf. Fig. 23)
- textual and visual specification of constraints (cf. Fig. 24). Vertices, edges and groups of them can be selected and assigned constraints.
- an extensible set of predefined local constraints (cf. Fig. 24)
- perspective 3-D views with real-time interactive graph exploration, e.g. navigation, zoom, camera flights (cf. Fig. 26-27)

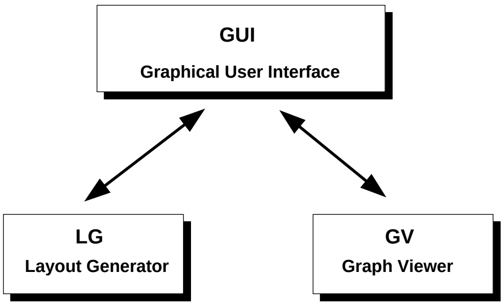

Figure 22: The GOLD architecture.

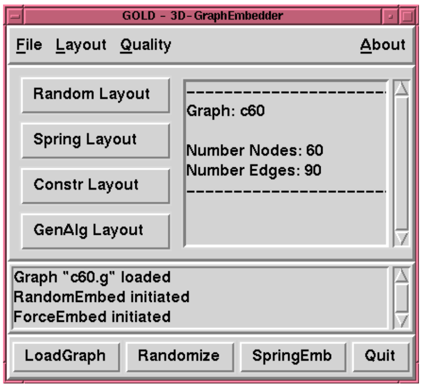

Figure 23: The GOLDSPEC Main Window.

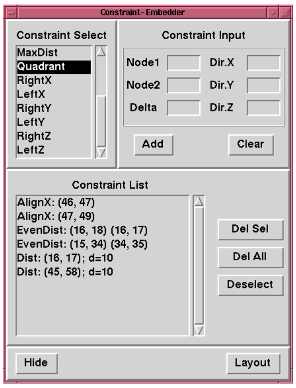

Figure 24: The Constraint Embedder User Interface.

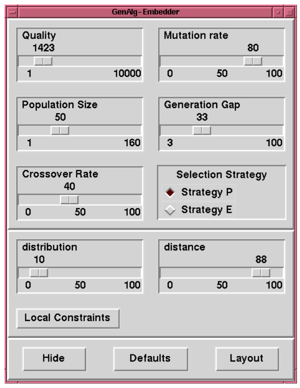

Figure 25: The Genetic Algorithm User Interface. Constraints can be specified with a toolbox similar to Fig. 24.

The GOLD architecture is comprised of three modules, each of which is realized as a separate process (cf. Fig. 22). The *graphical user interface* (GUI) module, written in Tcl/Tk [22], does just what its name says. The main window of the GUI module is shown in Fig. 23. This is also the master process controlling the other two.

The *layout generator* (LG) module, which is written in C++, offers several strategies for drawing graphs, among them random layout, the GEM-3D spring embedder algorithm [2], the local constraint propagation algorithm described in Sect. 2.1 and the genetic algorithm introduced in Sect. 2.2. It is easily extensible due to its flexible interface.

The *graph visualization* (GV) module is based on GeomView [23], an interactive geometry viewer, that can be embedded in applications using its interpreted LISP-like command language. This tools supports the 3-D representations of graph layouts in real-time. Additional navigation mechanisms for interactive exploration of the layout scene include rotation, translation and zoom (cf. Fig. 26).

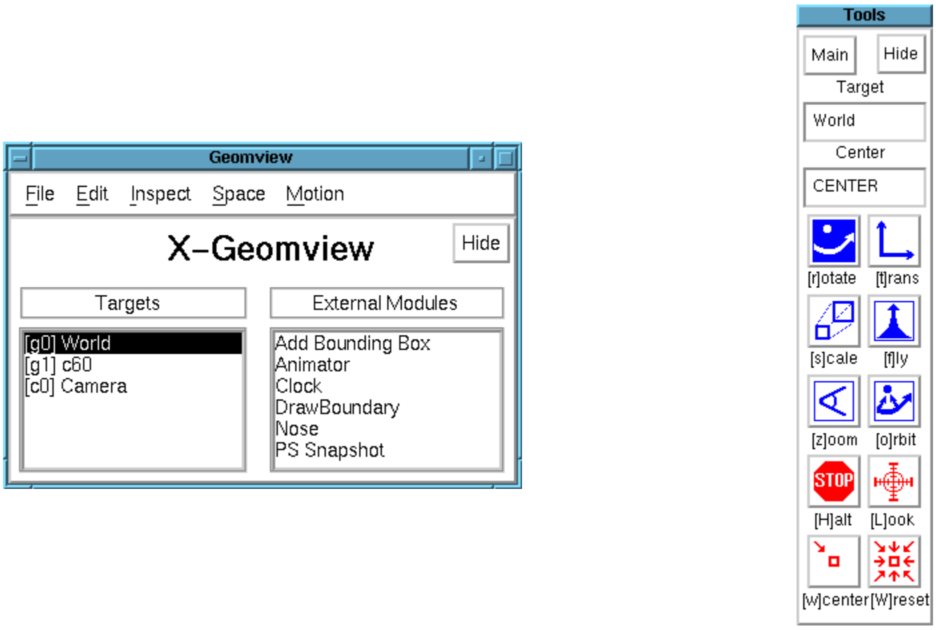

Figure 26: Geomview Main and Navigation tools window.

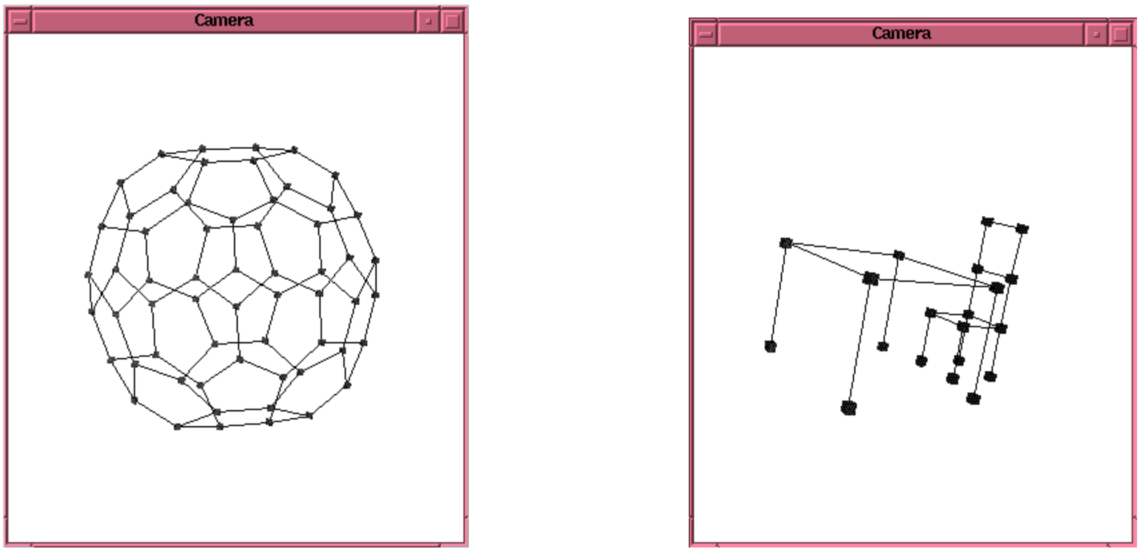

Figure 27: Two Geomview camera views on GOLD layouts.