Computational and Applied Mathematics, I. C. Brezinski and U. Kulish (Editors)
Elsevier Science Publishers B.V. (North-Holland) pp. 93–100 (1992). ISBN 0-444-89701-1.

Using genetic algorithms for planarization problems

F. Comellas

Departament de Matemàtica Aplicada i Telemàtica, Universitat Politècnica de Catalunya
ETSE Telecomunicació, Girona Salgado s/n, E-08034 Barcelona, Catalonia, Spain

Abstract

Two near-optimum planarization algorithms are presented. The algorithms belong to a general class of algorithms known as genetic algorithms because the search procedures on which they are based are inspired by the mechanics of natural selection and natural genetics. The first algorithm is intended to generate a near-maximal planar subgraph from a given graph and also provides the routing information needed to embed the subgraph on the plane. The second planarization algorithm studied finds a near-maximum independent set of a circle graph. This algorithm may be used for finding a routing on one layer from a set of n two-pin nets in a channel. After small modifications, it may also be used to predict the secondary structure of ribonucleic acids. The main advantage of both algorithms is that they are easily implemented in a parallel computer.

1. COMBINATORIAL OPTIMIZATION

An instance of a combinatorial optimization problem consists of a discrete set of solutions or *state space*, together with a *cost* or *objective function* that assigns a real number to each solution. The problem is to find a solution for which the cost function is optimal. Many optimization problems arising in practice are NP-complete: obtaining an optimal solution requires an exponentially increasing number of steps as the problems become larger. In this case several approaches are used to find good solutions. For particular NP-complete problems heuristic methods have been developed to give an acceptable answer in a reasonable time, although the optimum answer is not guaranteed. Heuristics are problem-specific. A heuristic procedure may be very efficient for finding near-optimal solutions for one NP-complete problem and useless for another. Also, most heuristic algorithms are descent algorithms with respect to the cost function. Consequently they are unable to escape local minima. Heuristic strategies come in several styles: constructive, divide-and-conquer methods and iterative improvements. In the first, the answer is built up directly. The second style divides the problem into subproblems of manageable size and then solves the subproblems. The solutions found must be adequately patched back together. This method is effective if the subproblems are disjoint. Iterative improvement is of more interest than other heuristics because of its broad range of applicability. Iterative improvement strategies attempt to generate from some existing suboptimal solution a better lower-cost solution. This solution then becomes the new configuration of the system, and the process is continued until no further improvements can be found. Standard iterative improvement is a *downhill* method. Each iteration moves the system to a configuration downhill from the previous one. The system usually becomes trapped in local minima. In practice, it is customary to carry out the process from many random initial configurations and save the best solution found. For very large problems this process is computationally expensive and there are still no guarantees of finding the desired solution.

Simulated annealing (SA) may be seen as a modification of iterative improvement which allows the system to move uphill in a controlled manner. The SA method comes from the analogy made between the states of a physical system, e.g. a liquid, and the configurations of a system in a combinatorial optimization problem. If the temperature of the interacting molecules in a liquid is suddenly reduced below its freezing point, the result will be a disordered glassy state with an energy higher than the true crystalline ground state. In fact the molecules are in a local energy minimum. On the other hand, if the temperature of the liquid is reduced slowly (annealing), waiting for equilibrium to be reached before a new reduction is made, the liquid freezes to the solid state through a cooling process that leads to the crystalline state, which is the global energy minimum. In the analogy with the combinatorial optimization problem the parameters being varied are equated with atomic positions in the liquid and its energy is identified with the cost function being optimized. The temperature is then defined as a control parameter related to the probability that changes which make the state worst will be accepted. This ensures a more exhaustive search of the state space. In the algorithm, a change of state that decreases the energy is always accepted; if the energy increases, the change is accepted with a certain probability that depends on the temperature of the system, according to the rule $e^{-\Delta E/T}$. At a given temperature several exchanges are attempted; then the process is repeated after decreasing the temperature. The system is gradually cooled until it is stopped according to some criteria such as when the number of changes accepted is small and/or the reduction of the energy is not significant. The number of attempts made at a given temperature has to be large enough to obtain a good statistical set of trials. The algorithm may also take into account the number of successful attempts at each temperature to decide whether the state space has been properly searched, and the search may be finished for this temperature. The SA technique has been successfully applied to scheduling problems like the traveling salesman problem, spatial organization problems like the chip placement and several other problems (see [1,2]). However, SA suffers from one major drawback: It requires careful tuning of its control parameters to achieve good results.

Neural networks based on the Hopfield model (see [3]) have been used with success for solving different combinatorial problems. These networks are composed of many simple computing elements (or *artificial neurons*) which cooperatively search the state space to find a local or global maximum or minimum.

Another family of algorithms with a possible broad range of applicability are genetic algorithms. Genetic algorithms (GA) were first introduced by J.H. Holland in the 60's, and have been successfully applied to the travelling salesman problem, pattern recognition, classifier systems, pipeline operations, scheduling, symbolic system evolution, and some other problems (see [4] for an extensive description and bibliography).

In a genetic algorithm the starting point is always a collection, known as *population*, of possible solutions generated at random. A suitable encoding of each solution in the population is used in order to compute its *fitness*. At each iteration a new population, or *generation*, is obtained by *mating* the best of the old solutions with one another. To create the next generation, new solutions are formed through *reproduction*, *crossover* and *mutation*. The solutions that will be considered for crossover are probabilistically selected according to the fitness values from the set that constitutes the current generation. This new population become the *parent pool*. Usually, a constant number of solutions are selected so that the maintained population is of fixed size. Crossover creates two new *child* solutions from two solutions sampled from the parent pool. In this way, fitter parents have a better chance of producing children. This is done for the whole population. Children solutions are obtained by interchanging random parts of their parents. Some randomness is also introduced through a mechanism called *mutation* to ensure that the algorithms avoid getting stuck at local minima. Mutation changes selected parts of a solution without keeping the original. The crossover and mutation operations are done with probabilites p_{cros} and p_{mut}. This ensures that some solutions from the current generation will be kept in the new generation. Once a new generation is created, the fitness of all solutions is evaluated and the process is repeated. At each generation the best solution is recorded. The algorithm ends when the results stabilize or the optimal solution, when it can be identified, is reached.

In this paper we will discuss the applicability of genetic algorithms to some NP-complete problems that arise in Graph Theory. Other methods such as neural nets and simulated annealing have been previously considered for dealing with these problems (see [5]). For all of them genetic algorithms might be of interest. There follows a list of several of these problems:

- *Max (or min) cut problem.* Consists in finding, for an edge weighted graph $G(V, E)$, a partition of V, $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$, such that the sum of the weights corresponding to the edges joining both sets is maximal (or minimal). The graph partitioning problem corresponds to the particular min cut case in which all weights are equal.
- *Independent set problem.* This problem consists in finding an independent set of maximal size $V' \subseteq V$ such that between any two vertices in V' there is no edge.
- *Graph colouring problem.* This is to find a minimal coloring of a graph $G(V, E)$, i.e. a set of l colors and a mapping of V to this set such that any two adjacent vertices have a different color and the set of colors has minimal cardinality.
- *Steiner tree problem.* Given an undirected connected and weighted graph $G(V, E)$ and a proper subset V' of V, find a minimum-weight tree which spans the vertices of V' and, if necessary, some others.
- *Planar subgraphs.* Given a (non planar) graph, find a maximal planar subgraph.

In this work genetic algorithms are used to solve two different graph planarization problems. In Section 2 we describe a genetic algorithm that finds a near-maximal planar subgraph from a given graph and also provides the routing information needed to embed the subgraph on the plane. Section 3 presents another planarization algorithm that finds a near-maximum independent set of a circle graph. After small modifications, it may also be used to predict the secondary structure of ribonucleic acids. Finally, in Section 4 the conclusions are presented.

2. GRAPH PLANARIZATION

The first graph planarization problem studied is directly related to the design of printed circuit boards and the routing of very large scale integration circuits (VLSI), and consists in finding a near-maximal planar subgraph from a given graph, in general non-planar. Jayakumar *et al.* [6] proposed an $O(N^2)$ near-optimal planarization algorithm, where N is the number of vertices. For the same problem, Takefuji and Lee [7], used an $N \times N$ neural network. The genetic algorithm presented here, as in the case of the Takefuji and Lee algorithm, not only yields this near-maximal subgraph but also provides the routing information for the embedding on the plane of the subgraph founnd. Both algorithms are able to find a new maximal planar subgraph with 20 edges instead of 19 from the nonplanar graph with 10 vertices and 22 edges which Jayakumar *et al.* used as an example in [6]. For solving this problem using a genetic algorithm, the state space of possible solutions is obtained from the original graph by drawing its vertices in a single row. A connection between two vertices, when considered, is made by either an upper edge or a lower edge. Edge crossings may thus appear. The algorithm tries to find a drawing without crossings and with the maximum possible number of edges from the original graph. Let us supose that the given graph has M edges. We code each possible solution by a list of M elements. Each position in the list corresponds to an edge of the original graph, and has values -1, 0 or 1 according to whether, for this solution the edge is a lower edge, is not considered or is an upper edge. In the first generation we generate at random a population of P states from the given graph to be planarized. The crossover on pairs of solutions $x^{\mathrm{par}} = x_1 x_2 \ldots x_M$ and $y^{\mathrm{par}} = y_1 y_2 \ldots y_M$, with $x_i, y_i \in -1, 0, 1$, is done by randomly choosing a cutting point $r < M$ and creating the child lists as follows: $x^{\mathrm{chld}} = x_1 x_2 \ldots x_r y_{r+1} \ldots y_M$ and $y^{\mathrm{chld}} = y_1 y_2 \ldots y_r x_{r+1} \ldots x_M$. Mutation is done by replacing the value of one list element randomly chosen with another admissible value generated at random. The fitness of a solution is evaluated according to the equation:

$$f(x) = n_c v_c + (M - n_e) v_e$$

Where n_c is the total number of crossings in solution x, n_e is the number of edges from the original graph considered in this solution and v_c and v_e the values assigned repectively to a crossing and to an edge. The existence of a crossing between two upper edges (or two lower edges) is easy to determine from the single row representation used.Two edges (i,j) and (l,m) cross if $i < l < j < m$ or $l < i < m < j$. Figure 1 shows a run of the genetic algorithm for a random generated graph with 12 vertices and 28 edges. A population of 200 individuals was considered. Other parameters were set as follows: $p_{\mathrm{cross}} = 0.9$, $p_{\mathrm{mut}} = 0.3$, $v_c = 1.1$ and $v_e = 1$. Figure 2 shows the initial graph and Figure 3 shows the best planar subgraph found at generation 120. The same set of parameters with a population of 80 individuals was used for finding, at generation 38, the maximal planar subgraph with 20 edges which contradicts the result of [6].

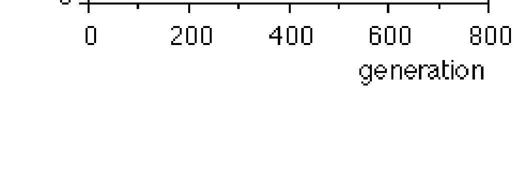

Figure 1. Evolution of the average fitness and best generated solution.

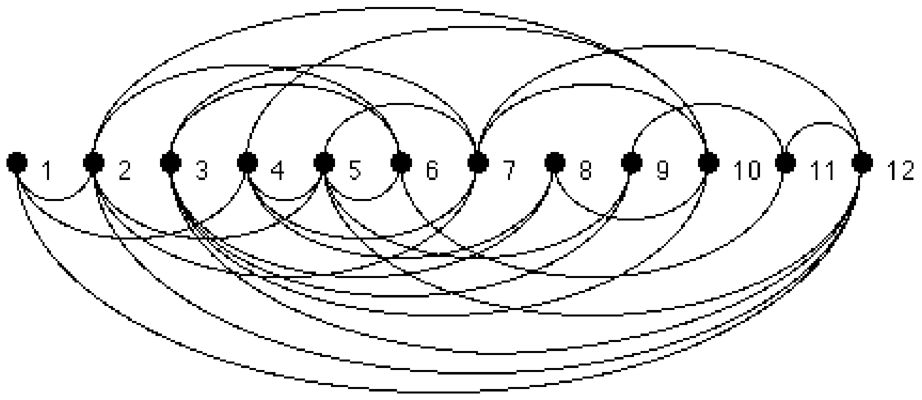

Figure 2. The initial non planar graph.

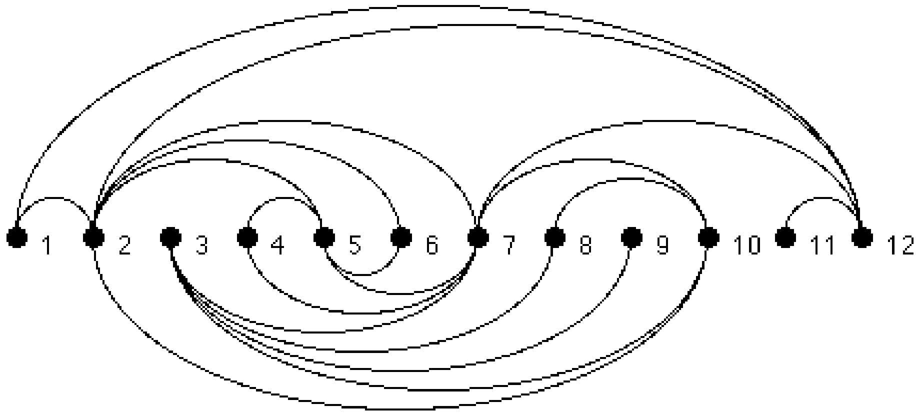

Figure 3. A planar subgraph found at generation 120.

2. PLANARIZATION OF CIRCLE GRAPHS

The second algorithm presented here finds a near-maximum independent set of a circle graph. An independent set in a given graph is a set of vertices, no two of which are adjacent. An independent set with the largest possible cardinality is a maximum independent set of the graph. A circle graph is the graph associated to a finite set of chords of a circle in such a way that each vertex of the graph corresponds to a chord and that there is an edge between each pair of vertices whose corresponding chords intersect (see Figure 4).

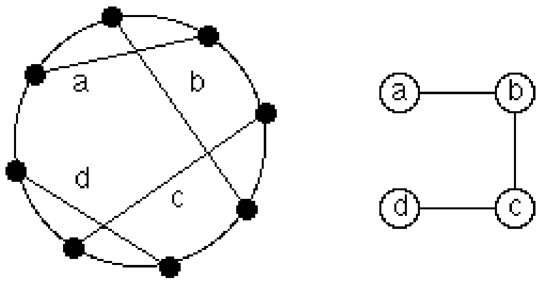

Figure 4. A set of chords and its circle graph.

Although the problem of finding a maximum independent set for arbitrary graphs is NP-complete [8], several polynomial time algorithms have been developed for finding a near-maximum independent set in a circle graph [9,10]. Neural nets have also been considered for this problem (see [11]).

As in the previous genetic algorithm , for this algorithm we code each possible solution by a list of M elements (M is the number of chords of the circle graph). Each position in the list corresponds to a chord of the original graph, and has values 1 or 0 according to whether or not the chord is considered for this solution. The algorithm follows the same steps as the previous one but uses a different cost function. In this case the cost function also depends on the chord length, and the fitness of a solution is evaluated according to the equation:

$$f(x) = L - \sum_{\text{present}} l_i + \sum_{\text{crossing}} l_i$$

Where L is the sum of the length of all chords in the circle graph to be planarized and l_i is the length of chord i. Figure 5 shows a run of the genetic algorithm for a circle graph with 10 vertices and 20 edges. A population of 200 individuals was considered. and the crossing and mutation probabilities were $p_{\text{cross}} = 0.9$, $p_{\text{mut}} = 0.3$. Figure 6 shows the best graphs found at generation 1. Figure 7 shows the best graphs found at generation 125.

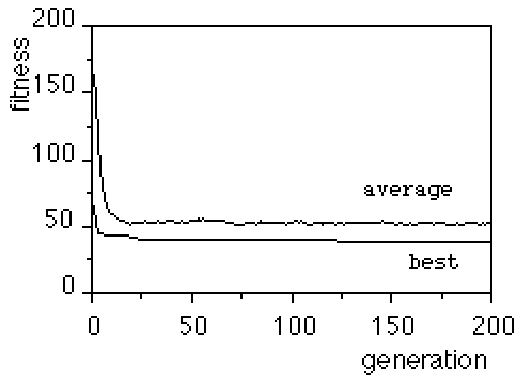

Figure 5. Evolution of the average fitness and best generated solution (circle graph planarization).

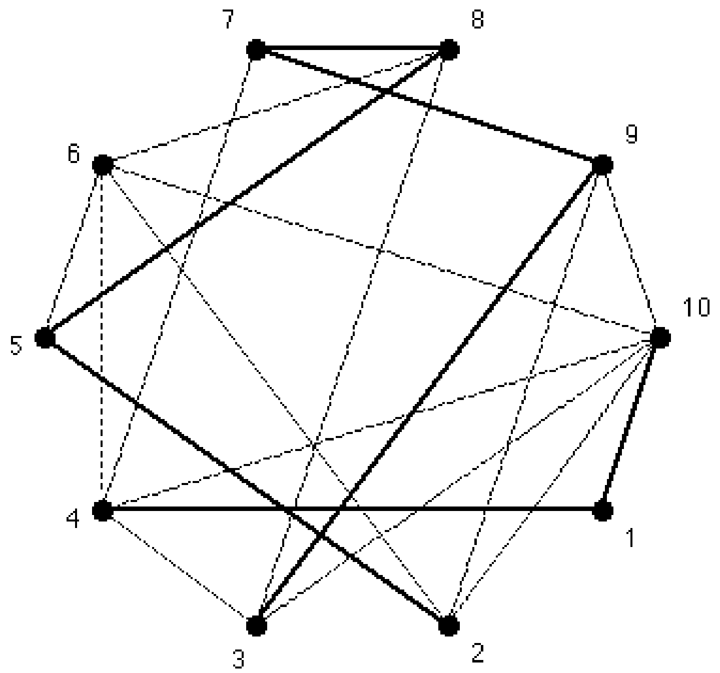

Figure 6. Best graph at generation 1.

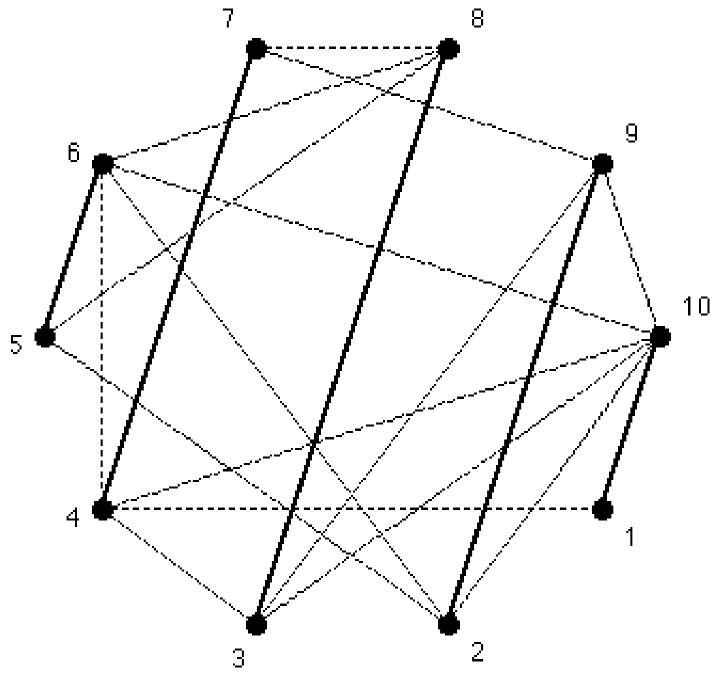

Figure 7. Best graph found at generation 125.

A modification of the algorithm based on [11] is useful for predicting the secondary structure of ribonucleic acids (RNA). The primary structure of RNA is determined by the sequence of organic bases. The folding of the chains into a two-dimensional shape determines the secondary structure. Non intersecting edges in a circle graph may be related with the base pairs for this folding. To generate a stable RNA structure here it is required to maximize the number of non intersecting edges or base pairs. The use of the RNA structure stability model from Tinoco *et al.* [12], enables the computation of the stability number of the resulting structures. This number is also used in the cost function. The modification was implemented and we were able to reproduce the result of Takefuji *et al.* [11] for the sequence of 38 bases used by them with their neural net. Work is still in progress for adapting the algorithm presented to process bigger sequences.

4. CONCLUSIONS

The results show that the genetic algorithms presented in this work perform well, and that with a similar (or less) computational effort than in other approaches (neural networks, simulated annealing, etc.) it is possible to obtain the desired solutions.

The main advantage of the method lies in its simplicity and the fact that by their very nature the algorithms may be easily implemented on a parallel computer.

5. REFERENCES

1 S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by Simulated Annealing, Science, 220 (1983), 671-680.

2 E. Aarts and J. Korst, Simulated Annealing and Boltzmann Machines, John Wiley & Sons, Chichester, 1989.

3 J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, Biol. Cybernet., 52 (1985), 141-152.

4 D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

5 R.H.J.M. Otten and L.P.P.P. van Ginneken, The Annealing Algorithm, Kluwer Academic Publishers, Boston, 1989.

6 R. Jayakumar, K. Thulasiraman and M.N.S. Swamy, $O(n^2)$ algorithms for graph planarization, IEEE Trans. Computer-Aided Design, 8 (1989), 257-267.

7 Y. Takefuji and K.C. Lee, A near-optimum parallel planarization algorithm, Science, 245 (1989), 1221-1223.

8 M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

9 F. Gavril, Algorithms on circular-arc graphs, Networks, 16 (1986), 357-369.

10 K. J. Supowit, Finding a maximum planar subset of a set of nets in a channel, IEEE Trans. Computer-Aided Design, 6 (1987), 93-94 .

11 Y. Takefuji, L.L. Chen, K.C. Lee and J. Huffman, Parallel Algorithms for finding a near-maximum independent set of a circle graph, IEEE Trans. Neural Nets., 1(3) (1990), 263-267.

12 I. Tinoco, O.C. Uhlenbeck and M.D. Levine, Estimation of secondary structure in ribonucleic acids, Nature, 230 (1971),362-367.