Recombination Operators for Evolutionary Graph Drawing

Daniel Kobler[1] and Andrea G. B. Tettamanzi[2]

[1] Department of Mathematics, Swiss Federal Institute of Technology,
EPFL, CH - 1015 Lausanne, Switzerland
E-mail: Daniel.Kobler@epfl.ch

[2] Università degli Studi di Milano, Dipartimento di Scienze dell'Informazione
Via Comelico 39, I-20135 Milano, Italy
E-mail: tettaman@dsi.unimi.it

Abstract. This paper illustrates an evolutionary algorithm for drawing graphs according to a number of esthetic criteria. Tests are carried out on three graphs of increasing difficulty and the performance of four different recombination operators are compared. The results are then briefly discussed and compared to those obtained with a tabu search.

1 Introduction

A number of data presentation problems involve the drawing of a graph on a two-dimensional surface, like a sheet of paper or a computer screen. Examples include circuit schematics, communication and public transportation networks, social relationships and software engineering diagrams. In almost all data presentation applications, the usefulness of a graph drawing depends on its readability, i.e. the capability of conveying the meaning of the diagram quickly and clearly. Readability issues are expressed by means of *esthetics,* which can be formulated as optimization criteria for the drawing algorithm [2].

An account of esthetic criteria that have been proposed and various heuristic methods for satisfying them can be found in [15]. An extensive annotated bibliography on algorithms for drawing graphs is given in [4] and [2]. The methods proposed vary according to the class of graphs for which they are intended and the esthetic criteria they take into account. For most reasonable esthetic requirements, however, it turns out that solving this problem exactly is prohibitively expensive for large graphs.

Evolutionary algorithms are a class of optimization methods inspired by Biology. It is assumed that the reader is already familiar with the main concepts and issues relevant to these algorithms; good reference books are [13, 8, 12, 1].

2 The Problem

Given an unoriented graph $G = (V, E)$, we want to determine the coordinates for all vertices in V on a plane so as to satisfy a certain number of esthetic requirements.

The esthetic criteria employed in this work are the following:

- there should be as few edge crossings as possible, ideally none;
- the length of each edge should be as close as possible to a parameter L;
- the angles between edges incident to the same vertex should be as uniform as possible (ideally the same).

A usually considered criterion requiring the vertices to be evenly distributed on the available space is in fact entailed by the second and third criteria stated above.

In general, the optimization problems associated with most esthetics are NP-hard [10, 11]. For instance, it has been proven that even just minimizing the number of edge crossings is an NP-hard problem [6]. Therefore the problem described above is also NP-hard, thus providing a valid motivation for resorting to an evolutionary approach.

The use of evolutionary algorithms for drawing *directed* graphs, a problem of great practical importance, already began to be explored by Michalewicz [12]. Surprisingly, however, that application disappeared from subsequent editions of his book. Michalewicz's work considers only two criteria, namely that arcs pointing upward should be avoided and there should be as few arc crossings as possible. Direction of edges is not addressed here. Rather, we deal with other types of graphs arising, for example, when representing on paper telecommunication networks, where links are always bidirectional.

3 The Algorithm

The algorithm whose elements are illustrated below is a generational replacement island-based evolutionary algorithm. In such an algorithm, there may be several populations, the *islands*, each of them undergoing evolution according to a generational replacement evolutionary algorithm, and from time to time good individuals migrate from an island to another.

3.1 Encoding and initialization

How a graph is drawn on a plane, i.e. a candidate solution to our problem, is completely determined by the (x, y) coordinates assigned to each vertex. Therefore, a genotype consists in a vector $((x_1, y_1), \ldots, (x_{|V|}, y_{|V|}))$, where (x_i, y_i) are the coordinates of the i-th vertex of the graph, encoded as two integers in $\{-4096, \ldots, 4095\}$, which can be considered as the basic *genes* of an individual. It is worth noticing that this is not a bit-string representation and thus genetic operators always act on pairs of coordinates (or vertex positions) according to their meaning.

Initial vertex positions for each individual are independently randomly generated and according to the same uniform probability over the whole drawing page. Generation of initial graphs could also be carried out with the help of greedy algorithms, which indeed have been tried, although they are not discussed in this paper.

3.2 Fitness

Three factors contribute to determining an individual's fitness, one for each esthetic criterion:

- the number of edge crossings, χ;
- the mean relative square error σ of edge lengths defined as

$$\sigma = \frac{1}{|E|} \sum_{e \in E} \left(\frac{\|e\| - L}{L} \right)^2, \tag{1}$$

  where $\|e\|$ is the length of edge e;
- the cumulative square deviation Δ of edge angles from their ideal values, defined as

$$\Delta = \sum_{v \in V} \sum_{k=1}^{N_v} \left(\psi_k(v) - \frac{2\pi}{N_v} \right)^2, \tag{2}$$

  where N_v is the number of edges incident into vertex v and the $\psi_k(v)$, $k = 1, \ldots, N_v$, are the angles between adjacent vertices.

Multi-objective techniques can be used for this problem, but in our approach, we decided to combine the objectives in one function. An individual's fitness, f, to be maximized, is then calculated as follows:

$$f = a \frac{1}{\sigma + 1} + b \frac{1}{\chi + 1} + c \frac{1}{\Delta + 1}, \tag{3}$$

where a, b and c are constants that control the relative importance of the three criteria and compensate for their different numerical magnitudes. Their values have been empirically determined as $a = 0.1$, $b = 0.8$ and $c = 0.1$ for the experiments described below; however, the reader should be aware that by modifying these constants the drawings produced by the algorithm can widely vary.

Since the three criteria employed conflict with each other and cannot in general be all completely satisfied at the same time, the fitness defined in (3) can only take the theoretical maximum of 1 in very few special cases.

3.3 Selection and crossover

Two selection strategies have been used: elitist fitness proportionate selection with a simple fitness scaling and ranking selection as described in [16].

Four different types of crossover, each needing two parents to create one or two offsprings, have been experimented. Crossover is applied with a given probability p_{cross} to each couple of selected individuals. In the first two crossovers, the two parents are equivalent:

- the *uniform* crossover, where the vertex positions that make up an individual's genotype have the same probability of being inherited from either parent;

- the *convex hull* crossover, where the coordinates of each vertex in the offspring are a linear combination of the coordinates of the same vertex in its parents: suppose that (x_1, y_1) and (x_2, y_2) are the coordinates of a vertex in the two parents, then the same vertex in the offspring will have coordinates (X, Y), where X and Y are two independent, normally distributed random variables with mean respectively $\frac{x_1+x_2}{2}$ and $\frac{y_1+y_2}{2}$ and with the same variance σ^2_{cross}, which is a parameter of the algorithm.

We also introduce two new crossovers, in which the first parent plays a more important role. In both cases, a subset V' of vertices is created and the offsprings are created as follows: the first offspring inherites the coordinates of a vertex v from the first (resp. second) parent if $v \in V'$ (resp. $v \notin V'$). The opposite holds for the second offspring. The two crossovers differ in their construction of V':

- the *plane cut* crossover, similar to one-point crossover, but applied to the grid on which the drawing is done. The line joining two randomly generated points splits the grid in two parts, say A and B. Then V' is the set of vertices that are in part A of the grid in the first parent.
- the *neighborhood* crossover, which tries to take advantage of good parts of a drawing. A set V' of given cardinality n' (parameter of the algorithm) is obtained by doing Breadth-First Scanning of the graph, starting from an arbitrary vertex (that is, take its neighbors, and then their neighbors, ...). In the case of a graph with several connected components, several starting vertices may have to be used in order to bring the cardinality of V' up to n'.

Overall, the algorithm is elitist, in the sense that the best individual in the population is always passed on unchanged to the next generation, without undergoing crossover or mutation.

3.4 Mutation and migration

Mutation perturbs individuals by adding to each of their vertices independent centered Gaussian noise, whose variance σ^2_{mut} is a parameter of the algorithm. This mutation operator is very similar to the convex hull crossover described above, the main difference being that in the case of crossover both parents participate in setting the mean of the new vertex position distribution.

If there are several islands, some migration occurs on a regular basis, defined by the user. Each island i sends to island $i+1$ (where the successor of the last island is island 1) a copy of its best individual currently available. This new individual replaces the worst individual in island $i+1$.

4 Experimental Results

The algorithm described above (denoted by EA) was implemented and run on several Pentium 120 and 200 workstations, under the Windows 95/NT operating systems and on Silicon Graphics workstations (200 MHz).

4.1 The Test Suite

The experiments were performed using three test graphs, that will be denoted H, D and N.

- H is the hypercube in 4 dimensions and contains $|V_H| = 16$ vertices and $|E_H| = 32$ edges.
- D represents the divisibility links of the integers from 2 to 30: the vertices corresponding to two numbers are linked if and only if one of these numbers divides the other. By removing the isolated vertices, this graph has $|V_D| = 25$ vertices and $|E_D| = 51$ edges.
- N is a piece of Internet. This graph is much bigger than the other two, having $|V_N| = 325$ vertices and $|E_N| = 327$ edges. It has several connected components.

The desired length for edges was set to $L = 8192/\sqrt{|V|}$, i.e. the average distance between closest vertices when they are uniformly distributed over the whole drawing plane.

4.2 Test Runs

A set of test runs was defined with the aim of understanding which combinations of parameter settings and operators work better. For each of the two graphs H and D, all combinations of the following alternatives were tried (between parentheses are given short codes for all alternatives):

- one island of 100 individuals ($\langle\text{popSize}\rangle = \texttt{C}$), two islands of 50 individuals ($\langle\text{popSize}\rangle = 2\texttt{L}$) or four islands of 25 individuals ($\langle\text{popSize}\rangle = 4\texttt{I}$);
- fitness proportionate selection or ranking selection ($\langle\text{selection}\rangle = \texttt{f}$ or $\texttt{r}$);
- $p_{\text{mut}} = 0.1$ ($\langle\text{mutRate}\rangle = \texttt{X}$) or $p_{\text{mut}} = 0.01$ ($\langle\text{mutRize}\rangle = \texttt{C}$);
- convex hull crossover with variance $\sigma^2_{\text{cross}} = L$ ($\langle\text{crossover}\rangle = \texttt{cl}$), neighborhood crossover with cardinality $n' = \lfloor |V|/2 \rfloor$ ($\langle\text{crossover}\rangle = \texttt{n}n'$), plane cut crossover ($\langle\text{crossover}\rangle = \texttt{p}$) or uniform crossover ($\langle\text{crossover}\rangle = \texttt{u}$);
- $p_{\text{cross}} = 0.5$ ($\langle\text{crossRate}\rangle = \texttt{H}$) or $p_{\text{cross}} = 0.2$ ($\langle\text{crossRate}\rangle = \texttt{V}$).

The mutation variance was always set to $\sigma^2_{\text{mut}} = L$. Furthermore, convex-hull crossover was always used without mutation, since it already carries a kind of mutation with it.

Experiment were code-named according to the following convention:

$$\langle\text{graph}\rangle\langle\text{popSize}\rangle\langle\text{selection}\rangle\texttt{ul}\langle\text{mutRate}\rangle\langle\text{crossover}\rangle\langle\text{crossRate}\rangle.$$

For graph N, which is considerably more demanding, only a few runs with one island, using the combinations that worked best on the other two graphs, were performed.

All runs were carried on for 10,000 generations, corresponding to an average execution time of about one hour for the two “simpler” graphs and of several hours for the “tougher” graph.

4.3 Results

Overall, 168 experimental runs were performed on both graphs H and D, whose results are given in Table 1. The best solution of every column is in bold. The results for graph N are summarized in Table 2 and Fig. 1 shows the best results, in terms of fitness, obtained by the EA applied to graphs D and N. Because of the large number of the possible parameter settings, these experiments do not allow robust assessments, but they give some tendencies.

A quick inspection of the results shows that, on average, runs with more islands tend to give better results, even though the difference with the single-island runs is not dramatic.

⟨pS⟩ = C	⟨pS⟩ = 2L	⟨pS⟩ = 4I	Run description	⟨pS⟩ = C	⟨pS⟩ = 2L	⟨pS⟩ = 4I
0.110	0.104	0.107	⟨pS⟩ful0clH	0.086	0.085	0.087
0.109	0.104	0.110	⟨pS⟩ful0clV	0.086	0.085	0.081
0.187	0.187	0.145	⟨pS⟩fulCn8H	0.144	0.208	0.173
0.187	0.187	0.179	⟨pS⟩fulCn8V	0.206	0.171	0.191
0.148	0.169	0.146	⟨pS⟩fulCpH	0.170	0.151	0.168
0.174	0.167	0.163	⟨pS⟩fulCpV	0.160	0.167	0.148
0.186	0.186	**0.187**	⟨pS⟩fulCuH	**0.225**	0.225	0.173
0.166	0.166	**0.187**	⟨pS⟩fulCuV	0.204	**0.226**	0.224
0.179	0.179	0.145	⟨pS⟩fulXn8H	0.224	0.166	0.204
0.165	0.165	0.178	⟨pS⟩fulXn8V	0.131	0.207	0.205
0.138	0.141	0.184	⟨pS⟩fulXpH	0.158	0.154	**0.226**
0.141	0.164	0.162	⟨pS⟩fulXpV	0.156	0.166	0.224
0.188	**0.188**	0.165	⟨pS⟩fulXuH	0.139	0.207	0.159
0.144	0.144	0.186	⟨pS⟩fulXuV	0.165	0.208	0.159
0.106	0.106	0.109	⟨pS⟩rul0clH	0.083	0.085	0.085
0.103	0.106	0.107	⟨pS⟩rul0clV	0.085	0.085	0.087
0.145	0.145	0.184	⟨pS⟩rulCn8H	0.167	0.156	0.207
0.181	0.181	0.140	⟨pS⟩rulCn8V	0.208	0.164	0.205
0.189	0.167	0.158	⟨pS⟩rulCpH	0.167	0.193	0.207
0.166	0.168	0.186	⟨pS⟩rulCpV	0.219	0.137	0.156
0.143	0.143	0.168	⟨pS⟩rulCuH	0.219	0.206	0.171
0.164	0.164	0.183	⟨pS⟩rulCuV	0.173	0.153	0.174
0.145	0.145	0.163	⟨pS⟩rulXn8H	0.127	0.151	0.221
0.180	0.180	0.145	⟨pS⟩rulXn8V	0.145	0.177	0.220
0.140	0.162	0.146	⟨pS⟩rulXpH	0.157	0.145	0.156
0.161	0.178	0.153	⟨pS⟩rulXpV	0.178	0.169	0.168
0.157	0.157	0.162	⟨pS⟩rulXuH	0.127	0.127	0.142
0.161	0.161	0.166	⟨pS⟩rulXuV	0.111	0.140	0.205
0.156	0.158	0.158	average	0.158	0.161	0.172

Table 1. Summary of results for graph H (on the left) and graph D (on the right) after 10,000 generations.

Run description	NCfulCn162H	NCfulCpH	NCfulCuH	NCfulCuV	NCfulXuH	NCfulXpH
f	0.091	**0.097**	0.089	0.088	0.020	0.047

Table 2. Summary of results for graph N after 10,000 generations.

A comparison of the effectiveness of the four proposed recombination operators can be carried out by calculating the average fitness reached by runs using each one of them:

Recombination	uniform	convex hull	plane cut	neighborhood
avg. f on H	**0.167**	0.107	0.161	**0.167**
avg. f on D	0.179	0.085	0.171	**0.182**

There is a clear indication that convex hull crossover is the worst of the four, and that uniform and neighborhood crossovers are the best, with a slight advantage for the latter. It is interesting to notice that these two recombination

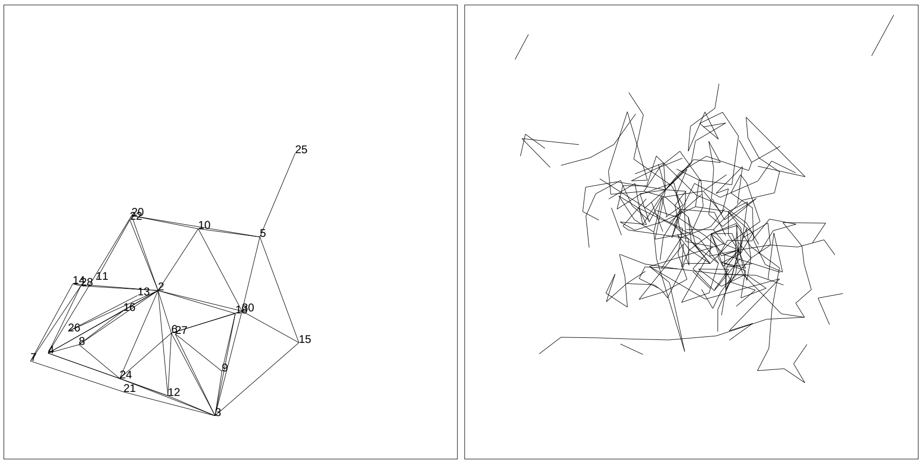

Fig. 1. Layouts of graph D (on the left) having fitness 0.226 ($\chi = 5$, $\sigma = 0.084$ and $\Delta = 168.204$) and of graph N (on the right) having fitness 0.097 ($\chi = 886$, $\sigma = 0.059$ and $\Delta = 1173.7$). Solutions found by the EA after 10,000 generations.

operators are the ones that, respectively, use the least and the most topological information about the graph. The higher crossover rate appears to be more advisable, although not by far.

As for mutation, runs with the lower mutation rate consistently gave better results. Of the two selection strategies employed, fitness-proportionate selection with scaling fares a little better than ranking selection, but this might change for different settings of the selection pressure in the latter.

4.4 Tests with a tabu search

In order to allow some comparisons with other results, a tabu search has also been designed to solve this problem. Tabu search (TS) will not be detailled here: the interested reader can refer to [7,9]. A solution s' is said to be a neighbor solution of s if s' can be obtained by applying a movement to s. Here a movement consists in adding a centered Gaussian noise to a random vertex v with variance σ^2_{TS}. Movements that bring v back in a square of side $\frac{\text{size of grid}}{500}$ centered on a previous position of v are forbidden during a certain number of iterations. This last number is regularly changed to a new value, randomly chosen within some bounds that depend on the mean length of the tabu list B_{length} as suggested in [5]. Such dynamic tabu lists have shown to be quite robust [5, 14]. The number of (randomly selected) movements M tested at each iteration of the TS is also given by the user.

A relatively small set of test runs has been defined for the tabu search, applied to graphs H and D. Similarly to Sect. 4.2, the experiments were code-named according to the following convention:

$$\langle\text{graph}\rangle\langle\sigma^2_{\mathrm{TS}}\rangle\langle\text{basicLength}\rangle\langle\text{mvtNb}\rangle$$

Run description	f
H500DC	**0.189**
H500CC	**0.189**
H500CL	0.188
H1000DC	0.187
H1000CC	0.188
H1000CL	0.188
HLDC	0.186
HLCC	0.185
HLCL	0.180
H3000DC	0.181
H3000CC	0.181
H3000CL	0.176
H5000DC	0.176
H5000CC	0.176
H5000CL	0.172

Run description	f
D400DC	0.163
D400CC	0.149
D400CL	0.118
D800DC	0.158
D800CC	0.175
D800CL	0.155
DLDC	0.227
DLCC	0.227
DLCL	0.221
D2500DC	0.207
D2500CC	0.206
D2500CL	0.219
D4000DC	0.248
D4000CC	**0.249**
D4000CL	0.180

Run description	f
N110CC	0.102
NLDC	**0.105**
NLCC	0.104
N1150CC	0.103

Table 3. Summary of results for graphs H, D and N with tabu search.

where

- $\langle \sigma^2_{\mathrm{TS}} \rangle$ is a number or the length L;
- $\langle$basicLength$\rangle$ is `D` (if $B_{\mathrm{length}} = 500$) or `C` (if $B_{\mathrm{length}} = 100$);
- $\langle$mvtNb$\rangle$ is `C` (if $M = 100$) or `L` (if $M = 50$).

The number of iterations is such that the number of solutions evaluated by the TS is equal to the number of individuals generated by the EA. Since evaluating a neighbor solution in the TS can be done faster than calculating the fitness of an individual, the execution time is divided by a factor of 2 or 3 for graphs H and D (even more for graph N).

The results are given in Table 3, the best solution for each graph being shown in bold. The values of σ^2_{TS} taken for graphs H and D are different, in order to have in both cases (approximately) the values $L/4$, $L/2$, L, $1.5 \cdot L$ and $2.5 \cdot L$. We notice that, as expected, the exact value of B_{length} is not important. Also, it is better to visit 100 neighbor solutions at each of the 10,000 iterations than 50 neighbor solutions during 20,000 iterations. But the value of M must not be increased too much, since otherwise the number of iterations would be too small for the algorithm to "have time" to find a good solution.

Using the best solutions obtained by the EA and the TS as initial solution for a tabu search with a small value of σ^2_{TS} (and less iterations), as a kind of "post-optimisation", did not provide significant improvements.

5 Discussion

If we compare the results obtained by the EA and the TS, we notice that the latter gave better values for f. Moreover, even if both algorithms evaluate the same number of solutions, the TS needs less time to do so. The definition of a movement used in the TS was chosen to make a more meaningful comparison with the EA, but other definitions may be considered as well.

The performance of the EA can be increased by using a directed mutation operator. One which has been tried with good results moves a vertex in a new position which is normally distributed around the center of the positions of neighbor vertices, with a variance inversely proportional to the number of these

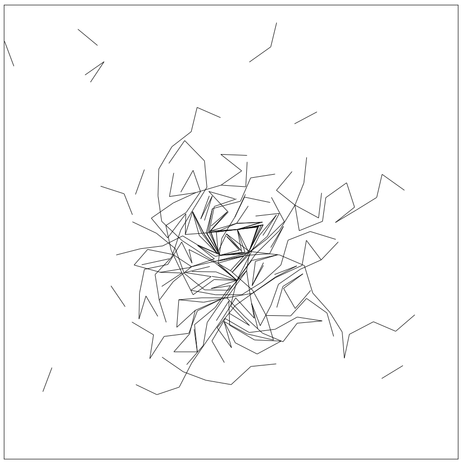

Fig. 2. A layout of graph N found by the tabu search after 10,000 iterations, having fitness 0.105 ($\chi = 138$, $\sigma = 0.008$ and $\Delta = 1364.58$).

neighbors. This operator has not been included in this paper for lack of space and time, but it will be experimented within a future work. Its use in the TS has not yet been tried.

The drawings obtained for the graph N by both algorithms (see Fig. 1 and 2) show that the quality of the solution can easily be improved by moving some vertices. But these improvements (easily detected by a human observator) do mainly concern the angles between edges and would therefore, because of the high value of Δ, have only a small effect on the value of f.

Even though one might think that using evolutionary algorithms for graph drawing represents a departure from classical algorithmic strategies found in computational geometry literature, a number of implicit similarities with well-known classical heuristics can be found. For instance, including a term inversely proportional to σ as defined in (1) in the fitness function recalls the so-called *spring embedder* force-directed method [3], whereby the drawing process is to simulate a mechanical system, where vertices are replaced by rings and edges are replaced by springs: the springs attract the rings if they are too far apart and repel them if they are too close.

Since the settings of the parameters controlling the relative weights of the esthetic criteria are very arbitrary and the quality of a drawing is inherently subjective, an idea for further work would be to allow the algorithm to interact with a human operator, using its responses to optimally tune the parameters relevant to the fitness function.

Acknowledgments

The research of the first author was supported by a Swiss National Fund for Scientific Research project under grant 21-45070.95. The research of the second author was partially supported by M.U.R.S.T. 60% funds. These supports are gratefully acknowledged. The authors thank Prof. Gianni Degli Antoni for his support and useful discussions and Genetica—Advanced Software Architectures S.r.l. in Milan for the extensive use of their computing equipment.

References

1. T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, Oxford, 1996.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. Technical report, Available on the Internet, URL: `ftp://wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.ps.Z`, 1989.
3. P. Eades. A heuristics for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
4. P. Eades and R. Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS-89-09, Department of Computer Science, Brown University, 1989.
5. C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63, 1996.
6. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.
7. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publ., 1997.
8. D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, 1989.
9. A. Hertz, E. Taillard, and D. de Werra. Tabu search. In J. K. Lenstra, editor, *Local Search in Combinatorial Optimization.* Wiley, 1995.
10. D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3(1):89–99, 1982.
11. D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 5(2):147–160, 1984.
12. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1992.
13. H.-P. Schwefel. *Numerical optimization of computer models.* Wiley, Chichester; New York, 1981.
14. E. Taillard. *Recherches itératives dirigées parallèles.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1993.
15. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):61–79, 1988.
16. D. Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, CA, 1989. Morgan Kaufmann.