An Evolutionary Heuristic for the Maximum Independent Set Problem

Thomas Bäck
University of Dortmund
Department of Computer Science
Systems Analysis Research Group, LS XI
D–44221 Dortmund, Germany
baeck@ls11.informatik.uni-dortmund.de

Sami Khuri
San José State University
Department of Mathematics & Computer Science
One Washington Square
San José, CA 95192–0103, U.S.A.
khuri@sjsumcs.sjsu.edu

***Abstract*— The results obtained from the application of a genetic algorithm, GENEsYs, to the NP-complete maximum independent set problem are reported in this work. In contrast to many other genetic algorithm based approaches that use domain-specific knowledge, the approach presented here relies on a graded penalty term component of the fitness function to penalize infeasible solutions. The method is applied to several large problem instances of the maximum independent set problem. The results clearly indicate that genetic algorithms can be successfully used as heuristics for finding good approximative solutions for this highly constrained optimization problem.**

I. Introduction

Once the NP-hardness of a combinatorial optimization problem is established, the search for an optimal solution is abandoned. The goal then becomes one of finding a good heuristic, i.e. a polynomial running time algorithm that can find solutions close to the optimal. In most cases, traditional heuristics are problem dependent; a heuristic is tailored to the specific problem it is trying to solve.

In this work, we present an alternative approach that uses *genetic algorithms* as a generalized heuristic for solving NP-hard combinatorial optimization problems. The application of a genetic algorithm is demonstrated here for the *maximum independent set problem*. These algorithms have been successfully applied to a broad range of problems. This wide range can be tackled by genetic algorithms mainly due to the fact that they work with an encoding of the domain rather than with the problem domain itself. The interested reader is refered to chapter 5 of [5] for more applications.

This robustness concerning the application domain is achieved by working with the coding of the parameter set rather than with the input data itself, such that genetic algorithms are nowadays used in a variety of problem domains.

The outline of the paper is as follows: Section II gives a short introduction to genetic algorithms. In section III, the maximum independent set problem and its representation for an application of the genetic algorithm are explained, and section IV presents the experimental results.

II. Genetic Algorithms

Genetic algorithms (GAs) [7; 9] are the best known representative of a class of direct random search algorithms based on the model of organic evolution, so-called evolutionary algorithms (see e.g. [17]). Canonical genetic algorithms represent the individuals (search points) of a population as binary vectors $\vec{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ of fixed length n. A fitness function $f : \{0,1\}^n \rightarrow I\!R$ that characterizes the actual optimization problem provides the quality measure for individuals. The fitness values are used by the selection procedure to direct the search towards regions of higher fitness, hopefully leading to an optimal solution.

The classical probabilistic proportional selection operator uses the relative fitness $p_s(\vec{x}_i) = f(\vec{x}_i) / \sum_{j=1}^{\mu} f(\vec{x}_j)$ to serve as selection probabilities (μ denotes the population size). In case of minimization tasks or negative fitness values $f(\vec{x}_i)$ is usually linearly transformed before calculating selection probabilities. This technique is known as linear dynamic scaling (see [7], pp. 123–124, or [8]).

Innovation, i.e. new information, is introduced into the population by means of mutation, which works by inverting bits with a small probability p_m (e.g. $p_m \approx 0.001$ [10]). Though mutation is often interpreted as a rather unimportant operator in genetic algorithms [9], recent theoretical work gives strong evidence for an appropriate choice of a mutation rate $p_m = 1/n$ on many problems [2; 12].

The recombination (crossover) operator allows for the exchange of information between different individuals. The original one-point crossover [9] works on two parent individuals (which are randomly chosen from the population) by choosing a crossover point $\chi \in \{1, \ldots, n-1\}$ at random and exchanging all bits after the χ^{th} one between both individuals. The crossover rate p_c (e.g., $p_c \approx 0.6$ [10]) determines the probability per individual to undergo crossover. The crossover operator can be extended to a generalized multi-point crossover [10] or even to uniform crossover, where an operator randomly decides for each bit whether to exchange it or not [19]. The strong mixing effect introduced by uniform crossover is sometimes helpful to overcome local optima.

After a random initialization of the population, the genetic algorithm proceeds by iterating the steps fitness evaluation, selection, recombination, and mutation until a termination criterion is fulfilled. Usually, the algorithm is terminated after a predefined number of iterations of the basic cycle, and the best individual in the final population serves as the result of the optimization process.

For the experiments reported in section IV the genetic algorithm software package GENEsYs is used [1]. This implementation is based on the widely used GENESIS software by Grefenstette (see [3], pp. 374–377), but allows for

The research of Thomas Bäck is supported by grant Schw 361/5–2 from Deutsche Forschungsgemeinschaft (DFG).

more flexibility concerning genetic operators and data monitoring. The parameter settings for our experiments are given in section IV, where the experimental results are presented. First, however, section III presents an introduction to the maximum independent set problem and the representation of solution candidates as binary strings.

III. The Maximum Independent Set Problem

The maximum independent set problem consists of finding the largest subset of vertices of a graph such that none of these vertices are connected by an edge (i.e., all vertices are independent of each other). Thus, if $G = (V, E)$ denotes a graph where V is the set of nodes and E the set of edges, the problem is to determine a set $V' \subseteq V$ such that $\forall i, j \in V'$ the edge $\langle i, j\rangle \notin E$ and $|V'|$ is maximum. This problem is NP-complete (see [6], pp. 53–56).

In the following, we present a formal definition of the maximum independent set problem by making use of Stinson's terminology for combinatorial optimization problems [18]:

Problem instance: A graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges. An edge between vertices i, j is denoted by the pair $\langle i, j\rangle \in E$, and we define the *adjacency matrix* (e_{ij}) according to

$$e_{ij} = \begin{cases} 1 & , \quad \text{if } \langle i, j\rangle \in E \\ 0 & , \quad \text{otherwise} \end{cases} .$$

Feasible solution: A set V' of nodes such that $\forall i, j \in V' : \langle i, j\rangle \notin E$ (i.e., $e_{ij} = 0$).

Objective function: The size $|V'|$ of the independent set V'.

Optimal solution: An independent set V' that maximizes $|V'|$.

In order to encode the problem to use a genetic algorithm, we choose the following representation of a candidate solution as a binary string $(x_1, x_2, \ldots, x_n)$: $x_i = 1 \Leftrightarrow i \in V'$. This way, the i^{th} bit indicates the presence ($x_i = 1$) or absence ($x_i = 0$) of vertex i in the candidate solution. Note that a particular bitstring may (and will often happen to) represent an infeasible solution. Instead of trying to prevent this, we allow infeasible strings to join the population and use a penalty function approach to guide the search towards the feasible region [13]. The penalty term in the objective function has to be graded in the sense that the farther away from feasibility the string is, the larger its penalty term should be. The exact nature of the penalty function, however, is not of high importance if it fulfills the property of being graded (see [16]).

Taking this design rule for a penalty function into consideration, we developed the following fitness function to be maximized by the genetic algorithm:

$$f(\vec{x}) \quad = \quad \sum_{i=1}^{n} \left(x_i - n \cdot x_i \cdot \sum_{j=i}^{n} x_j e_{ij} \right) \quad . \tag{1}$$

This fitness function penalizes infeasible strings $\vec{x}$ by a penalty of n for every node j in the candidate solution V' represented by $\vec{x}$ that is connected to a node $i \in V'$. For feasible strings $\vec{x}$, $f(\vec{x}) \geq 0$ and the fitness value is just given by the number of nodes in the independent set represented by $\vec{x}$.

In solving the maximum independent set problem, we also have a solution for two other graph problems: The *minimum vertex cover problem* (given $G = (V, E)$, find the smallest subset $V' \subseteq V$ such that $\forall \langle i, j\rangle \in E : i \in V' \vee j \in V'$) and the *maximum clique problem* (given $G = (V, E)$, find the largest subset $V' \subseteq V$ such that $\forall i, j \in V' : \langle i, j\rangle \in E$). The close relationship between these problems is characterized by the following lemma (see e.g. [6]):

Lemma 1
For any graph $G = (V, E)$ and $V' \subseteq V$, the following statements are equivalent:

- V' is the maximum independent set in G.
- $V - V'$ is the minimum vertex cover of G.
- $V - V'$ is the maximum clique in $G^C = (V, E^C)$, where $E^C = \{ \langle i, j\rangle \mid i, j \in V \wedge \langle i, j\rangle \notin E \}$.

Consequently, one can obtain a solution of the minimum vertex cover problem by taking the complement of the solution to the maximum independent set problem. A solution to the maximum clique problem is obtained by applying the maximum independent set heuristic to $G^C = (V, E^C)$.

IV. Experimental Results

The experiments reported in this section are performed by using a genetic algorithm with a population size $\mu = 50$, a mutation rate $p_m = 1/n$, crossover rate $p_c = 0.6$, proportional selection, and two-point crossover (which is, according to the experimental results reported in [4; 15], expected to perform better than the traditional one-point crossover). In order to become applicable to the maximum independent set problem, no component of this general genetic algorithm — except, of course, the fitness function — has to be modified. This fact reflects the wide applicability and robustness of genetic algorithms in contrast to problem-specific heuristics.

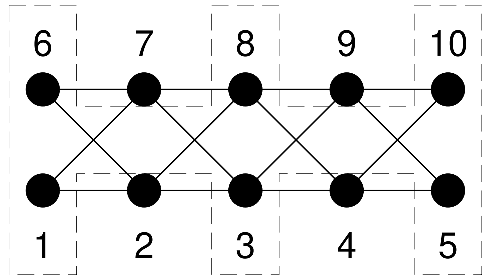

Fig. 1: Example graph "misp10" with $n = 10$ nodes. The independent set $V' = \{1, 3, 5, 6, 8, 10\}$, represented by the bitstring (1010110101), is indicated by the dashed lines. Notice that the independent set $\{2, 4, 7, 9\}$, represented by the bitstring (0101001010), is a local maximum.

In order to obtain large test problems for an applica-

misp102		misp100-01		misp100-02		misp100-03		misp100-04		misp100-05	
$f_{2\cdot 10^4}(\vec{x})$	N	$f_{2\cdot 10^4}(\vec{x})$	N	$f_{2\cdot 10^4}(\vec{x})$	N	$f_{2\cdot 10^4}(\vec{x})$	N	$f_{2\cdot 10^4}(\vec{x})$	N	$f_{2\cdot 10^4}(\vec{x})$	N
52	—	47	1	45	34	45	77	45	96	45	99
50	1	46	1	43	3	44	1	33	1	17	1
48	14	45	3	41	2	41	6	32	1		
46	32	44	6	40	3	37	3	25	1		
44	40	43	4	39	10	36	3	10	1		
42	10	42	4	38	1	34	1				
40	3	41	9	37	8	33	1				
		40	4	36	1	32	1				
		39	6	35	1	30	1				
		38	12	34	6	29	1				
		37	5	33	6	26	1				
		< 37	45	< 33	25	< 26	4				
$\bar{f} = 44.94$		$\bar{f} = 37.39$		$\bar{f} = 37.25$		$\bar{f} = 42.38$		$\bar{f} = 44.20$		$\bar{f} = 44.72$	

Table 1: Experimental results for the regular graph "misp102" with $n = 102$ vertices and five random graphs with edge density $d = 0.1$ ("misp100-01"), $d = 0.2$ ("misp100-02"), $d = 0.3$ ("misp100-03"), $d = 0.4$ ("misp100-04") and $d = 0.5$ ("misp100-05"). An independent set size $k = 45$ was chosen for the random graphs, but for the graph with $d = 0.1$ the genetic algorithm identified a larger independent set.

tion of the genetic algorithm to the maximum independent set problem, we make use of the scalable graph shown in figure 1, which can be constructed for an even number of nodes n ($n \geq 6$). If n is a multiple of 4, two equivalent global maxima of fitness value $|V'| = n/2$ are obtained by partitioning the set of vertices into those of even (respectively odd) node numbers. Otherwise, the unique global maximum is given by $V' = \{1, 3, \ldots, n/2, n/2+1,\ n/2+3, \ldots, n\}$, with fitness value $n/2+1$, and a local maximum is obtained from $V - V'$ with fitness value $n/2-1$. For $n = 10$, the corresponding bitstrings are $\vec{x}' = (1010110101)$ and its inverted form (0101001010) (see figure 1).

In addition to this graph, which has a highly regular structure, we use randomly constructed graphs which are created according to the following algorithm with input $k \in \{1, \ldots, n\}$ (number of nodes in V') and $d \in [0, 1]$ (edge density of the graph):

randomly select $V' = \{i_1, \ldots, i_k\} \subseteq V = \{1, \ldots, n\}$
for $i = 1$ **to** n **do**
  for $j = i + 1$ **to** n **do**
    if $((Random(0,1) < d)$ **and**
      $((i \notin V')$ **or** $(j \notin V')))$
    then $e_{ij} = 1$
    else $e_{ij} = 0$

The algorithm at random preselects k nodes $i_1, \ldots, i_k$ that are guaranteed to form an independent set (the graph may, however, contain different larger independent sets by chance, especially when the edge density is low). Edges are placed at random, according to the density parameter d, such that it is guaranteed that a member of V' is never connected to another member of V' (note that, according to the construction method, only loop-free graphs are generated).

For the experimental test regular graphs of size $n = 102$, respectively $n = 202$ (with a maximum independent set of size 52, respectively 102) and random graphs with $n = 100$, $k = 45$, respectively $n = 200$, $k = 90$ and $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ are used. For each of these problems, a total of $N = 100$ runs of the genetic algorithm are performed. These runs are evaluated according to the number of runs that yield solutions of identical quality. The results are summarized in table 1 (for the graphs with 102 respectively 100 vertices) and table 2 (for the graphs with 202 respectively 200 vertices) for the best results that were encountered during the 100 runs for each test problem. For each experiment, the average final best fitness value $\bar{f}$ over all 100 runs is indicated at the bottom of the table. The total number of function evaluations performed for each single run is indicated as an index t in the notation $f_t(\vec{x})$; for $n = 100$ we use a value of $t = 2 \cdot 10^4$, while this is doubled for $n = 200$. Consequently, only a small fraction of the search space (about $1.6 \cdot 10^{-24}\%$ for $n = 100$ respectively $2.5 \cdot 10^{-54}\%$ for $n = 200$) is tested by the genetic algorithm.

For the regular graphs "misp102" and "misp202", none of the runs of the genetic algorithm identified the globally optimal solution of quality 52 respectively 102, but for all runs a solution quality between 40 and 50 respectively 82 and 96 is obtained, i.e., solutions close to the optimal one are found. Finding the global optimum in case of these regular graphs becomes an extremely difficult problem due to large Hamming distances between local optima of similar quality (e.g. consider $f(101001010101001010) = 8$, $f(101010101101010101) = 10$, and the Hamming distance between both strings is 10).

A comparison of the results for the random graphs re-

misp202		misp200-01		misp200-02		misp200-03		misp200-04		misp200-05	
$f_{4\cdot 10^4}(\vec{x})$	N	$f_{4\cdot 10^4}(\vec{x})$	N	$f_{4\cdot 10^4}(\vec{x})$	N	$f_{4\cdot 10^4}(\vec{x})$	N	$f_{4\cdot 10^4}(\vec{x})$	N	$f_{4\cdot 10^4}(\vec{x})$	N
102	—	90	4	90	54	90	93	90	100	90	100
96	3	88	1	89	1	72	1				
94	3	85	2	80	4	70	2				
92	11	84	3	79	6	65	1				
90	33	82	2	78	3	62	2				
88	30	81	2	77	5	51	1				
86	12	80	4	75	4						
84	5	79	1	74	2						
82	3	78	5	73	3						
		77	5	71	1						
		76	1	70	1						
		< 76	70	< 70	16						
$\bar{f} = 88.90$		$\bar{f} = 68.75$		$\bar{f} = 81.05$		$\bar{f} = 88.22$		$\bar{f} = 90.00$		$\bar{f} = 90.00$	

Table 2: Experimental results for the regular graph "misp202" with $n = 202$ vertices and five random graphs with edge density $d = 0.1$ ("misp200-01"), $d = 0.2$ ("misp200-02"), $d = 0.3$ ("misp200-03"), $d = 0.4$ ("misp200-04") and $d = 0.5$ ("misp200-05"). An independent set size $k = 90$ was chosen for the random graphs.

veals that the edge density is the major factor which determines the complexity of the maximum independent set problem. The smaller (larger) the edge density, the fewer (more) runs succeed in finding a solution of quality $k = 45$ respectively $k = 90$ or better (which is possible in case of small edge density, e.g. for $d = 0.1$). For small edge density, the number of local optima grows due to the possibility of exchanges of groups of vertices and the existence of isolated vertices. As the edge density increases to a value of 0.5, the frequency of runs that identify the solution with 45 respectively 90 vertices grows steadily. For the smaller graphs, the genetic algorithm always found the best solution for an edge density above $d = 0.5$, while this property holds for the larger graphs already for $d = 0.4$.

Notice that, according to the construction mechanism, the edge density of the regular graph amounts to $\frac{4\cdot(n-2)}{n\cdot(n-1)} \approx 4/n$ (the regular graph has $2n-4$ edges, and the maximum number of edges is $n\cdot(n-1)/2$ if no loops are permitted). ¿From the experience with random graphs, it is clear that this small value provides further evidence for the complexity of the regular graph problems.

All runs of the genetic algorithm are characterized by the following properties, independently of the problem instance the algorithm is applied to: The initial phase of the search is used for finding feasible solutions from a completely infeasible initial popluation. The genetic algorithm quickly succeeds in leaving the infeasible region in each of the runs reported here, thus demonstrating the appropriateness of our graded panalty function approach. After at most 200 respectively 400 generations ($1 \cdot 10^4$ respectively $2 \cdot 10^4$ function evaluations) each run has settled in a local optimum and does not show further improvement. The quality of the optima found, however, clarifies the genetic algorithms' reliability for identifying good approximative solutions for the maximum independent set problem.

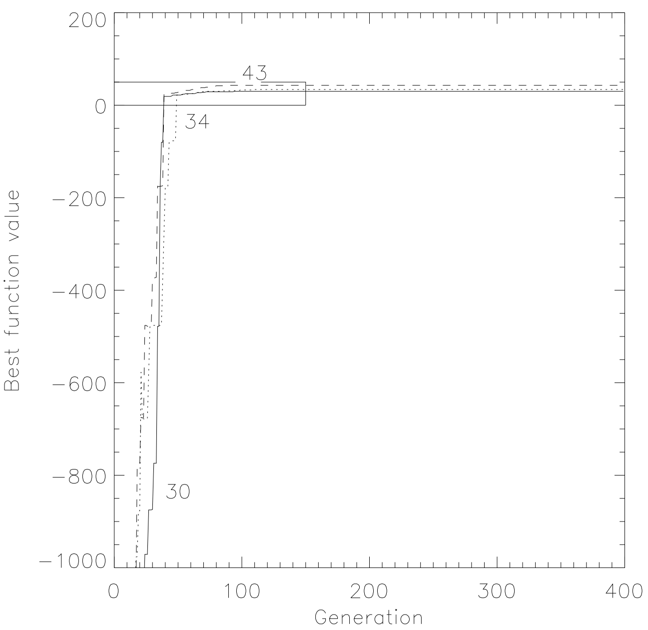

Fig. 2: Some representative courses of evolution for the maximum independent set problem (using the "misp100-01" example).

To illustrate the typical behavior of genetic algorithm runs, figure 2 shows the course of evolution for three different runs on the "misp100-01" problem. The best fitness value that occurred in the population is plotted over the generation number for each of the three runs. Each run is labeled by its final solution quality, and the ordinate axis is restricted to a smaller range of values than really observed (initially best fitness values are found around $-3.5 \cdot 10^3$). Note that only about 50 generations are required to enter the feasible region (which corresponds with nonnegative fitness values). Further progress is observed until approximately generation 100, and afterwards the search stagnates in local optima.

Figure 3 shows a magnification of the marked region from figure 2. This closer look reveals that between generations 50 and 100 a steady period of further improvement of feasible solutions takes place. During this stage of the search, the algorithm fine-tunes solutions towards one of the local optima of the search space.

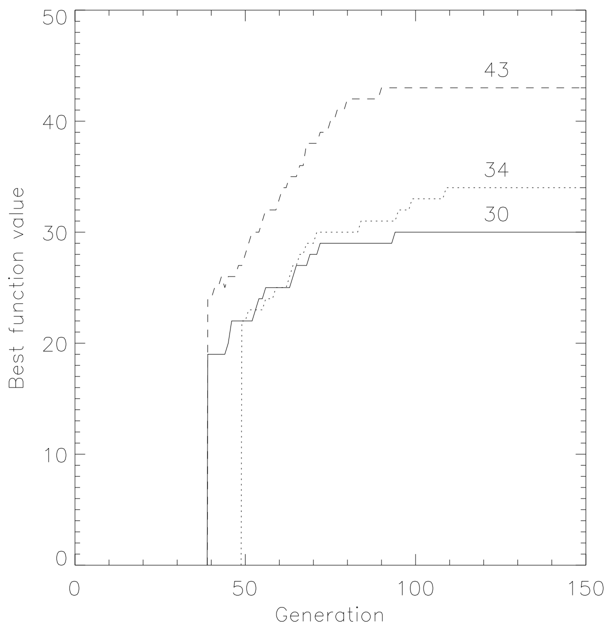

Fig. 3: Magnification of the marked region in figure 2.

V. Conclusion

We have shown in this work that genetic algorithms can be used in a fairly straightforward way to find good approximative solutions of the NP-hard maximum independent set problem. The robustness of our approach based on a graded penalty function for infeasible strings is demonstrated by the fact that no changes to the genetic algorithm are required. Thus, rather than having to construct tailored heuristics to handle the problem under consideration, we advocate the use of genetic algorithms where the only change to perform is the formulation of a new fitness function.

References

[1] Th. Bäck. GENEsYs 1.0. Software distribution and installation notes, Systems Analysis Research Group, LSXI, Department of Computer Science, University of Dortmund, Germany, July 1992. (Available via anonymous ftp to `lumpi.informatik.uni-dortmund.de` as file GENEsYs-1.0.tar.Z in /pub/GA/src).

[2] Th. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In Männer and Manderick [11], pages 85–94.

[3] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[4] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer. Biases in the crossover landscape. In Schaffer [14], pages 10–19.

[5] S. Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP–Completeness.* Freemann & Co., San Francisco, CA, 1979.

[7] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.

[8] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC–16(1):122–128, 1986.

[9] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[10] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.

[11] R. Männer and B. Manderick, editors. *Parallel Problem Solving from Nature 2.* Elsevier, Amsterdam, 1992.

[12] H. Mühlenbein. How genetic algorithms really work: I. mutation and hillclimbing. In Männer and Manderick [11], pages 15–25.

[13] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In Schaffer [14], pages 191–197.

[14] J. D. Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[15] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In Schaffer [14], pages 51–60.

[16] A. E. Smith and D. M. Tate. Genetic optimization using a penalty function. In Forrest [5], pages 499–505.

[17] W. M. Spears, K. A. De Jong, Th. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Machine Learning: ECML-93*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 442–459. Springer, Berlin, 1993.

[18] D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms.* The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 2nd edition, 1987.

[19] G. Syswerda. Uniform crossover in genetic algorithms. In Schaffer [14], pages 2–9.