Journal of Heuristics, 5, 145–158 (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

A Taxonomy of Evolutionary Algorithms in Combinatorial Optimization

PATRICE CALÉGARI, GIOVANNI CORAY, ALAIN HERTZ,* DANIEL KOBLER AND PIERRE KUONEN
Swiss Federal Institute of Technology, Department of Mathematics and Computer Science, CH-1015 Lausanne, Switzerland
email: Patrice.Calegari@epfl.ch, Giovanni.Coray@epfl.ch, Alain.Hertz@epfl.ch, Daniel.Kobler@epfl.ch, Pierre.Kuonen@epfl.ch

Abstract

This paper shows how evolutionary algorithms can be described in a concise, yet comprehensive and accurate way. A classification scheme is introduced and presented in a tabular form called TEA (Table of Evolutionary Algorithms). It distinguishes between different classes of evolutionary algorithms (e.g., genetic algorithms, ant systems) by enumerating the fundamental ingredients of each of these algorithms. At the end, possible uses of the TEA are illustrated on classical evolutionary algorithms.

Key Words: evolutionary algorithms, genetic algorithms, taxonomy

Introduction

One of the goals of our research project is to extract the fundamental ingredients of well-known Evolutionary Algorithms (EA), such as Genetic Algorithms (GA), scatter search (SS) and ant systems (AS), in order to study the role of these ingredients and their importance. Another goal of the project will be to take this study as a basis in order to analyze the best ways of implementing such algorithms on parallel machines.

This paper presents an attempt to classify EAs. A new scheme relying on a table, called the *Table of Evolutionary Algorithms* (TEA), is introduced. The TEA does not provide, nor does it replace, algorithm pseudo-codes. It merely informs about the algorithm's key elements. The primary goal of the TEA is to compare the principles of EAs, as opposed to comparing their performances. It should never be forgotten that its aim is not to explain the details of a given algorithm. It may however be used to describe algorithm classes, or to compare the characteristics of two algorithms a priori considered to be different. The paper is organized as follows. Section 2 introduces the main ingredients of EAs and Section 3 shows how these ingredients can be interpreted at different levels. Then, Section 3.2 describes the TEA classification. In Section 4 some very typical EAs are described using the TEA. At the end, concluding Section 5 discusses possible evolution trends for the proposed classification scheme.

*Author to whom all correspondence should be addressed.

1. Background and motivation

At the beginning of EA history, there was no ambiguity about what GAs were (Holland (1975)). Later, however, different ingredients were added to enhance GAs' performances, leading to algorithms which substantially differ from their original principles (Davis (1991), Goldberg (1989)). These algorithms are still often named GAs. Moreover, it is common to find a GA described with the same pseudo-code as an EA in the literature (Heitkötter and Beasley (1997)). Although the difference between these two classes of algorithms is usually explained, it seems that the distinction is often not clear. One of the risks such a situation leads to is that identical things are made several times under different names. This is the reason why, despite the existing ad hoc tutorials, a systematic means of describing the main ingredients of EAs in a short-hand way is a challenging task to investigate.

An example of the risk mentioned above is given by scatter search and GAs. As explained in (Glover (1994)), a number of evolutions of GAs used for solving optimization problem were basic elements of the initial scatter search framework, and have therefore been 'rediscovered'.

In the literature, works very often focus on the efficiency, or the utility, of some kind of operators (like crossover in GA for example (Spears and DeJong (1991), Syswerda (1989))). In our opinion, the important points concerning the structure of the algorithms are too often ignored. This could be schematized by saying that the interest is more brought to the implementation of the EAs than to the mechanisms of the algorithm itself. For example, in order to understand the 'philosophy' of an algorithm, the fact of using or not using a mutation operator in a GA may appear more important than the way it is done.

2. Main ingredients of an EA

A first step is to identify what are the fundamental ingredients that determine the characteristics of EAs.

2.1. Population

EAs were designed to solve combinatorial optimization problems. The fundamental difference between EAs and traditional methods, such as iterative or constructive algorithms, is that EAs handle a set of candidate solutions. In the context of EAs, this set is called a *population* and candidate solutions are called *individuals*. The population size can either be constant or change during its evolution. The result of an EA is the best individual ever met in the population. As for traditional combinatorial optimization algorithms, an objective function, called here *fitness function*, makes it possible to associate with each individual a *fitness value* that reflects its quality. Starting from an initial population of individuals (generated randomly or with a constructive algorithm), evolutionary methods try to improve the quality of the individuals by making the population evolve. The evolution is usually achieved using information exchanges between individuals in order to create new ones or to modify existing ones. Individuals that exchange information are called the *parents* and newly created or modified individuals are called the *offspring*. The exchange of information

is realized by operators that usually depend on the considered EA. In GA this is done by *crossover* techniques (Holland (1975)). Offspring is created using information coming from several individuals present in the current population. The number of parents participating to the creation of an offspring is an important ingredient of an EA since it defines how much information is merged at once. This number is simply called the *number of parents*. For example, in GAs the number of parent is constant and equal to 2, but there exist other EAs in which the number of parents can vary during the execution of the algorithm.

In addition to the parents, the creation of the offspring may also use some global information about the history of the population. This information represents the context in which the algorithm evolves, and is called *history of the population*. This context is generally handled by the population, and is updated with respect to the past of the population. Under the term 'history of the population' we consider information, taking into account the evolution, that cannot be gathered by looking at the current state of the population, but would need the historical account of the last generations. An example for this would be given by GAs in which the probability that is associated with each operator would be updated by taking into account the results obtained during the last couple of generations.

The *information sources* of an offspring are described by two ingredients, the number of parents and the history of the population (represented by the presence/absence of the abbreviation $h_{\text{Population}}$).

2.2. Neighborhood

An important question concerning the exchange of information between individuals is to determine which individuals are allowed to make exchanges. To answer that question, a neighborhood function can be associated with the population P. A neighborhood function is:

$$N : P \to \mathcal{P}(P)$$

where $\mathcal{P}(P)$ is the set of the subsets of P. The neighborhood function associates with each individual e a subset $N(e)$ of P called its neighborhood. This relationship must be symmetric: $e_1 \in N(e_2)$ implies $e_2 \in N(e_1)$. Such a neighborhood function can take the form of a graph in which a vertex is associated to each individual and there is an edge between two individuals e_1 and e_2 if $e_1 \in N(e_2)$.

An information exchange operator is applied to a subset $Q \subseteq P$ of parents such that $e_1 \in N(e_2)$ for each pair of individuals $\{e_1, e_2\}$ in Q. If $N(e) \cup \{e\} = P \; \forall e \in P$, the operator can be applied to any combination of individuals and P is said to be *unstructured*. Otherwise, the population is said to be *structured*. The most common structure is a grid.

2.3. Evolution of a population

Usually, the evolution of the population is achieved through a succession of evolution steps called *generations*. If the whole population can be changed from one generation to another,[1] an evolution step is said to be a *generational replacement*. If only a part of the population

is changed from a generation to another, the evolution is said to be *steady state*. With the use of parallel programming for the EAs, *asynchronous* evolution has appeared. In this last case, each individual is continuously changed without checking whether the others are also changed. For example in an asynchronous GA, the next individual that is to be replaced by an offspring can still be used between the beginning and the end of the creation of the offspring.

2.4. Solution encoding

There exist many ways to encode individuals. Even if chromosome-like strings are often used, the encoding method is mainly determined by the information that should be exchanged by individuals. The kind of information that is exchanged is on a higher level, whereas the way to do it (a crossover operator description, for example) may be considered on a lower level. The information exchange operator, and its coding, will be determined on the basis of the kind of information that has to be exchanged. However the kind of information to exchange in order to have an efficient EA depends on the problem considered. Indeed, once one has decided what information is important to exchange, the basic blocks of information, one can choose any encoding method and design the information exchange operator with respect to this encoding and the information blocks. Of course, some pairs (encoding, information exchange operator) may be very inefficient on a computer, but the choice of this pair is now only an implementation problem. Since encoding is highly problem-dependent,[2] this important feature will not be included in our general descriptive table.

Nevertheless, the encoding method has an impact on the behavior of an algorithm. It is important to describe these effects. Depending on the encoding method and the information exchange mechanism, newly created or modified individuals can represent infeasible solutions. An infeasible solution is a candidate solution that is not a solution to the considered problem. In such a case, individuals representing the infeasible solutions can be repaired, penalized, or killed. Individuals, that are repaired, are transformed so as to represent a solution to the problem. The fitness value of penalized individuals acquires a penalty that can depend on the distance between the candidate solution represented and a feasible solution. Individuals that are killed are simply deleted, or replaced by other new individuals. It should be noticed that some EAs can, by using convenient encoding and information exchange, avoid the creation of infeasible solutions.

2.5. Individual history

As explained in Section 2.1, information about the history of the population may be stored during the run of the algorithm. A similar information may exist at the individual level: each individual has some evolving information that does not concern the problem being solved but how the individual behaves in a certain situation (what is its mutation rate for example). This information is history at an individual level, called *history of the individual*, and is represented by $h_{\text{Individual}}$. Here again, the history covers information that cannot be determined by the current state of the individual.

Notice that this notion also applies to a generational replacement algorithm. Indeed, the history of a newly created individual is then defined on the basis of the history of its parents. An example of such a history is given by the Evolution Strategies (Bäck and Schwefel (1993)).

2.6. Individual improving

A way to bring significant improvement in the results obtained by an EA is to use hill-climbing or more advanced heuristic techniques at some stage of the computation (Glover (1977)). Some EAs thus apply a more or less sophisticated *improving algorithm* on newly created individuals. An improving algorithm is any change applied to a single individual, without using information of other individuals, in order to improve its fitness value. The improving algorithm can be a simple operation or a more sophisticated combinatorial algorithm (e.g., tabu search, simulated annealing).[3]

2.7. Noise

One of the major problem encountered with combinatorial algorithms is the premature convergence of the solution in a local optimum. In order to steer individuals away from local optima, EAs introduce some *noise* in the population. This noise can be generated by randomly perturbing some individuals, like the mutation operator does in a GA for example. The only requirement is that this noise has unexpected results on the fitness of an individual, in the sense that it does not necessarily improve it.

In the case of the scatter search approach (and its path relinking generalization), a somewhat different philosophy is used to produce new individuals—for the purpose of moving away from individuals that may cluster around local optima or some more complex region of attraction. Instead of using noise to produce unpredictable outcomes, a *diversification strategy* is used to systematically produce individuals that lie in new regions. The notion of diversification derives from the tabu search literature, where it is contrasted with randomization. Rather than seeking unpredictability, diversification seeks to attain an objective that implies a special form of order, such as maximizing a minimum weighted separation from chosen collections of points previously generated. (For a fuller discussion of the diversification concept, and its relation to the counterbalancing concept of intensification, see (Glover and Laguna (1997))).

2.8. The basic TEA

In order to be aware of the principles of an EA compared to another EA, the ingredients that characterize them must be easily readable. We therefore propose to create a one-row table that allows such comparisons. The main idea of the table is to have one column per ingredient developed in this section. At each position, an entry, that can be a number or an abbreviate information, gives the necessary indication for the corresponding criteria. Figure 1 shows such an empty table.

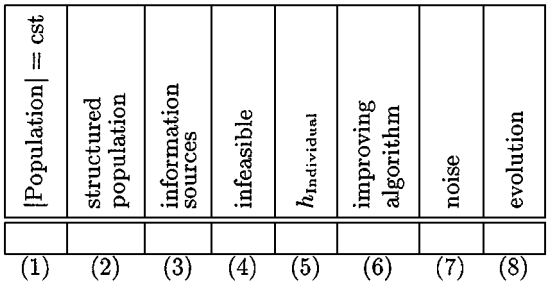

Figure 1. The basic TEA.

(1) a 'Yes' or a 'No', depending if the size of the population is constant or not.
(2) a 'Yes' or a 'No', depending if the population is structured or not.
(3) the number of parents for each offspring (nothing if this number is not fixed). The abbreviation h_P is added to the number if the history of the population is used.
(4) one of the four abbreviations: *nvr* (when infeasible individuals can never appear), *pen* (when the infeasible individuals are penalized), *rep* (when the infeasible individuals are repaired) or *die* (when the infeasible individuals are killed).
(5) a 'Yes' or a 'No', depending if the history of the individuals is used by the algorithm.
(6) a 'Yes' or a 'No', depending if an improving algorithm is applied to the individuals or not.
(7) a 'Yes' or a 'No', depending if noise is used or not.
(8) one of the three abbreviations: *gr* (when generational replacement is used), *ss* (when steady state is used) or *as* (when asynchronous mode is used).

For a first example, we use the very simple genetic algorithm described in Chapter 3 of (Goldberg (1989)). The following scheme summarizes this algorithm:

```
determine an initial population P_0 of p individuals (p even);
generation = 0;
repeat
    generation = generation + 1;
    while P_generation has less than p individuals
        select indi1 and indi2 in P_generation-1;
        (offsp1, offsp2)=crossover(indi1, indi2);
        put offsp1 and offsp2 in P_generation;
    endwhile
    for each individual in P_generation do
        mutation(individual);
    endfor
until generation ≥ max_generation.
```

The basic TEA associated with this algorithm is shown in figure 2.

|Population| = cst	structured population	information sources	infeasible	$h_{\text{Individual}}$	improving algorithm	noise	evolution
Yes	No	2	nvr	No	No	Yes	gr

Figure 2. The basic TEA for a simple genetic algorithm.

3. Hierarchical ingredients

3.1. Further description levels

Ingredients that were described in the previous section concern the individual level. Some other description levels may however be considered, in order to handle several populations, or even sets of populations. This section shows how the notions explained in Section 2 for the individual level can be understood at other description levels.

Usually, the notion of parents is only used when a new individual is created by combining other individual information. However, this notion can be generalized to any exchange of information. For example, consider the case where a population is split into sub-populations. A sub-population obtained by selecting a collection of individuals from two sub-populations I_1 and I_2 can be considered as the offspring, while I_1 and I_2 can be considered as the parents.

In fact, most of the ingredients of the previous section remain correct if 'individual' is replaced by 'sub-population'. With this in mind, the previous section can describe another level of an algorithm using sub-populations. In order to look at an EA in this manner, one must define what an element e is and in what sets S these elements are grouped. At the level considered, the EA works on a set of these elements. To illustrate what we mean, we take again the well-known GAs, and more specifically an island-based GA (IGA). In such an algorithm, at the lower level an element e is an individual and individuals are grouped to form *islands* (the sets S). Since the IGA works on each island independently, we can consider a level of islands where an element e is an island (the set S of the previous level) and where these islands are grouped to form an *archipelago* (the new set S). In the case of the IGA, only one archipelago is considered, but we could imagine a process that clusters archipelagi into '*meta-archipelagi*'. In such a case, we can still apply the same reasoning as for the previous level. Therefore, the previous section remains valid almost without modification for all levels. For instance, in the IGA case, the information exchange operator, corresponding to the crossover operator at the individual level, can be migration at the island level, as mentioned above. A definition for the 'fitness' of an island can be the mean fitness of the individuals in this island. Thus, an improving algorithm can improve this mean fitness (without using the other islands). The only ingredient of the previous section that cannot be easily generalized to the upper level, is the feasibility of an element: it is

not clear what an infeasible island can be. But the possibility is left for a suitable definition needed in future developed EAs.

Even if they do not exist yet (as far as we know), EAs can be imagined with even more levels. The IGA can for example be extended to a three-level algorithm, an archipelago-model GA, in which there are several archipelagi. Algorithms with more than two levels do not necessarily give better results, but they can enter in the above described framework.

A grouping concept whose function operates a bit differently from the function of islands derives from creating clusters of individuals that share particular features (Glover (1977)). This gives rise to strategies where 'within cluster' operations yield forms of intensification, while 'across cluster' operations yield forms of diversification.

Since, as explained in Section 2.1, an individual represents a candidate solution to the problem considered, we may say that the individual level is the basic level of an EA. In that case, the level in which an element is a set of individuals can be seen as a level above this basic level. More generally, if the elements e of a given level l are more elaborate than the elements e' of a level l', that is if the elements e' are components of the elements e, the level l can be considered as being higher than l'. But this can be extended to the other side of the basic level. Let us for example consider a set of chunks of information, called e', that describe an individual e through the 'state' of this set. This implies the use of a procedure to reconstruct a solution from the set of chunks (Rochat and Taillard (1995), Kuntz and Snyers (1994), Zufferey and Hertz (1997)). In addition to the evolution of the individuals e in a population, each of these sets of elements e' (one set for each individual) may also evolve as in an evolutionary algorithm. Thus the description of the EA may have a level where the sets S' of elements e' are individuals ($S' = e$).

3.2. The classification table

We are now ready to understand the complete classification table, based on the basic TEA introduced in Section 2.8. In this extended table, one row is filled per description level. An additional column is inserted on the left side in order to name the description level of each row. Figure 3 shows such a table with two description levels.

Column (0) names a description level by making explicit the set S of elements e concerned in the corresponding row. For example, at the lower level in a IGA, 'Island(Individual)'

$S(e)$ Set of elements e	$\|S\| = \text{cst}$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)

Figure 3. TEA: the table for evolutionary algorithm classification.

$S(e)$ Set of elements e	$\lvert S\rvert = \text{cst}$	...
Population(Individual)	Yes Δ	...

Δ The population size changes when...

Figure 4. Example of the use of the Δ-feature.

would mean that in the first row the elements *e* are the 'Individuals' grouped into 'Islands'.

The TEA is filled like explained in Section 2.8 for the basic TEA, except that we now consider sets and elements instead of populations and individuals. For example, columns (2) and (5) are now filled with:

(2) a 'Yes' or a 'No', depending if the set *S* is structured or not.
(5) a 'Yes' or a 'No', depending if the history of the elements *e* is used by the algorithm.

In the case the corresponding ingredient has no sense at a given level, a division contains the character '/'.

In standard EAs, the one-to-one relation between the levels of the algorithm and the rows of the table is true. But this table is more flexible: several rows are possible for a given level. For example, two different types of islands can be used and can be described by a row named 'Island1(Individual)' and a row named 'Island2(Individual)', grouped with 'Archipelago(Island1, Island2)'.

In order to improve the results, algorithms often use some kind of diversification in one of the ingredients we have presented. For example, one can imagine that the size of the population is constant most of the time, but that it is decreased from time to time and then brought back to its original value. If taken literally, one should put a 'No' in the column entitled "$|S| = \text{cst}$". But since, the overall idea is to have a constant population, we propose to associate 'Yes' with a special symbol Δ (for Diversification) in this column. Figure 4 shows how to describe a population whose size is decreased every now and then. How exactly the diversification is done can be commented beside the table.

4. Examples

This section presents five typical examples of EAs in pseudo-code, together with their TEA.

4.1. A classical genetic algorithm

We use again the simple genetic algorithm presented in Section 2.8, but consider the TEA in its standard shape. The TEA associated with this algorithm is shown in figure 5.

$S(e)$ Set of elements e	$\|S\| = \text{cst}$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
Population(Individual)	Yes	No	2	nvr	No	No	Yes	gr

Figure 5. A simple genetic algorithm.

4.2. An island-based genetic algorithm

The second example is a steady-state island-based GA, that uses migration every m generations:

```
determine k initial islands [P_0, ..., P_{k-1}];
generation = 0;
repeat
    generation = generation + 1;
    for each island i do
        select indi1 and indi2 in P_i;
        (offsp1, offsp2)=crossover(indi1, indi2);
        mutation(offsp1);
        improve(offsp1);
        mutation(offsp2);
        improve(offsp2);
        replace two individuals of P_i by offsp1 and offsp2;
    endfor
    if generation is multiple of m then
        for each island i do
            migrate the best individual of island P_i to island P_{(i+1) mod k};
        endfor
    endif
until stopping condition is met.
```

If we define the fitness of an island as the mean fitness of the individuals in this island, then the TEA associated with this algorithm is shown in figure 6.

4.3. A scatter search

The third example is a basic scatter search (Glover (1994)) that can be summarized by the following scheme:

$S(e)$ Set of elements e	$\|S\| = cst$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
Island(Individual)	Yes	No	2	nvr	No	Yes	Yes	ss
Archipelago(Island)	Yes	Yes	2	/	No	No	No	gr

Figure 6. An island-based genetic algorithm.

$S(e)$ Set of elements e	$\|S\| = cst$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
Population(Point)	Yes	No		rep	No	Yes	No	ss

Figure 7. A basic scatter search.

determine an initial set P_0 of points;
$i = 0$;
repeat
  $i = i + 1$;
  determine a set T_i of points by linear combinations of points in P_i;
  transform the points in T_i to get a set F_i of feasible solutions;
  improve the solutions in F_i to get a set D_i of points;
  select $|P_0|$ points in $P_{i-1} \cup D_i$ to form P_i;
until stopping condition is met.

The TEA associated with this algorithm is shown in figure 7. Notice that the position in TEA concerning the information sources is empty, since the number of parents used is not fixed.

4.4. An ant algorithm

We now consider an ant algorithm. We only give here a sketch of such an algorithm, but detailed description and definitions of the different terms can be found in (Colorni, Dorigo and Maniezzo (1991), Colorni, Dorigo, and Maniezzo (1992)):

$S(e)$ Set of elements e	$\|S\| = cst$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
Population(Ant)	Yes	/	0 h_S	nvr	No	No	No	gr

Figure 8. An ant algorithm.

initialise the trails;
cycle $=$ 0;
repeat
  cycle $=$ cycle $+$ 1;
  for each ant **do**
    construct a solution s_a using trails and visibility;
    evaluate the objective function at s_a;
  endfor
  update the trails;
until cycle $\geq$ max_cycle.

The corresponding TEA is shown in figure 8. Notice that the only information source for an ant is the history of the population (called trails in ant algorithms). Indeed, during the construction of a solution, an ant does not use the solutions provided by some given ants, but uses the objective function values obtained by the whole population during a certain number of cycles. Notice that the absence of parents in the information exchange operator leaves the notion of structured population undefined. This is why we have a '/' in the corresponding position of this TEA.

4.5. An emergent colonization algorithm

The last example illustrates the use of chunks of information. Consider the k-coloring problem where the vertices of a graph G have to be colored with a given number k of colors, so that the number of edges having both endpoints with the same color is minimized. The emergent colonization algorithm described in (Zufferey and Hertz (1997)) handles a population of agents located on vertices of G, each agent having a fixed color chosen in $\{1, \ldots, k\}$. At each iteration of the algorithm, two agents on adjacent vertices exchange their position, according to some rules. Given a population of agents, a coloring of G is obtained by assigning to each vertex v the color that is the most frequent among the agents on v. For more details, the reader is referred to (Zufferey and Hertz (1997)). The corresponding TEA is shown in figure 9.

5. Conclusion

We present a new approach to Evolutionary Algorithms with the purpose of highlighting the main distinguishing features of an EA and offering reflections about the way we see

$S(e)$ Set of elements e	$\|S\| = cst$	structured S	information sources	infeasible e	h_e	improving algorithm	noise	evolution
Population(Agents)	Yes	Yes	$2k\ h_S$	nvr	No	No	No	ss

Figure 9. An emerging colonization algorithm.

meta-heuristics in general, and EAs in particular. We introduced a table (TEA: Table of Evolutionary Algorithms) for the description of each typical algorithm. We illustrated how existing EAs can be characterized using our TEA. The TEA has been designed in order to be extensible to yet unknown classes of EAs. It makes explicit the hierarchical notion of 'level' in EAs; therefore, the TEA can also be helpful for the design of new classes of EAs. Indeed, by exploring many different ways to 'fill out' the TEA, one may discover new classes of EAs by their descriptive characterization. In the end it should be stressed that the TEA is not a static tool. In concert with the possible evolution of EAs, the TEA may evolve in order to take into account new ingredients that will eventually become popular in EAs.

Acknowledgments

The research presented in this paper were done in the framework of the project LEOPARD (parallel population-based methods for combinatorial optimization). It has been funded by the Swiss National Science Fund (# 21-45070.95/1). We wish to thank Fred Glover for his precious comments and advices on a previous version of this article, and also an anonymous referee for helpful comments.

Notes

1. With the possible exception of one or two individuals of the population.
2. But it could be introduced in a problem-specific table for EAs.
3. In such case, authors use the term of hybrid algorithms.

References

Bäck, Th. and H.-P. Schwefel. (1993). "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation* 1, 1–23.

Colorni, A., M. Dorigo, and V. Maniezzo. (1991). "Distributed Optimization by Ant Colonies." In MIT Press (ed.), *First European Conference on Artificial Life*, Bradford Books, pp. 134–142.

Colorni, A., M. Dorigo, and V. Maniezzo. (1992). "An Investigation of Some Properties of an Ant Algorithm." In R. Männer and B. Manderick (eds.), *Second European Conference on Parallel Problem Solving from Nature*. Brussels: Elsevier Publishing, pp. 509–520.

Davis, L. (1991). *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences* 8, 156–166.

Glover, F. (1994). "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing* 4, 131–140.

Glover and M. Laguna. (1997). *Tabu Search*. Norwell, MA: Kluwer Academic Publishers.

Goldberg, D. (1989). *Genetics Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley: Publishing Company.

Heitkötter, J. and D. Beasley. (1997). *The Hitch-Hiker's Guide to Evolutionary Computation (FAQ for comp.ai.genetic)*. URL:`ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html`.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Kuntz, P. and D. Snyers. (1994). "Emergent Colonization and Graph Partitioning," *Third International Conference of Adaptative Behavior*. MIT Press, pp. 494–500.

Rochat, Y. and E. Taillard. (1995). "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics* 1, 147–167.

Spears, W.M. and K. DeJong. (1991). "An Analysis of Multi-Point Crossover." In G.J.E. Rawlins (ed.), *Foundations of Genetic Algorithms.* Morgan Kaufmann, pp. 301–315.

Syswerda, G. (1989). "Uniform Crossover in Genetic Algorithms." In J.D. Schaffer (ed.), *Third International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 2–9.

Zufferey, N. and A. Hertz. (1997). "Coloration de graphes à l'aide de fourmis." Technical Report, EPFL, Lausanne, Switzerland.