Evolving Objects: a general purpose evolutionary computation library

M. Keijzer[1], J. J. Merelo[2], G. Romero[2], and M. Schoenauer[3]

[1] Danish Hydraulic Institute
mak@dhi.dk
[2] GeNeura Team, Depto. Arquitectura y Tecnología de Computadores
Universidad de Granada (Spain)
todos@geneura.ugr.es, http://geneura.ugr.es
[3] CNRS and École Polytechnique, France
marc@cmapx.polytechnique.fr

Abstract. This paper presents the *evolving objects library* (EOlib), an object-oriented framework for evolutionary computation (EC) that aims to provide a flexible set of classes to build EC applications. EOlib design objective is to be able to evolve any object in which fitness makes sense. In order to do so, EO concentrates on interfaces; any object can evolve if it is endowed with an interface to do so. In this paper, we describe what features an object must have in order to evolve, and some examples of how EO has been put to practice evolving neural networks, solutions to the Mastermind game, and other novel applications.

1 Introduction

Evolutionary Algorithms (EAs) are stochastic optimization algorithms based on a crude imitation of natural Darwinian evolution. They have recently become more and more popular across many different domains of research, and people coming from those "external" domains face a difficult dilemma: either they use an existing EA library, and then have to comply to its limitation, or write their own, which represent a huge work, and generally leads to ...some other limitations that their authors are not even aware of, mainly because these scientists are not closely related to recent EA research.

For instance, evolving any kind of objects, (e.g. Neural Networks), has been a difficult matter, mainly due to the lack of flexibility of current evolutionary computation libraries with respect to the representation used and the variation operators that can be used on that representation. Most libraries (such as [41, 42]; see [19] for a comprehensive list) allow only a few predefined representations.

Evolving other types of data structures hence often has to start by *flattening* them to one of the usual representations, such as a binary string, floating point array or LISP tree. In the case of NNs, for instance, this representation has to be decoded to evaluate the network (e.g. on a training set in the case of a regression problem), but it sometimes lacks precision (e.g. in the case of binary string representation), or expressive power: a string, whatever its shape, is a serialization of a complex data structure, and evolution of a string using standard string-based variation operators makes keeping actual building blocks together more difficult than the evolution of a structure more closely representing neural nets, such as two arrays of weights together with biases for 3-layer perceptrons, or, more generally, an array of objects representing . . . neurons.

Similarly, most existing libraries propose only a limited range of ways to apply Darwinian operators to a population (e.g. limited to some proportional selection and generational replacement), or/and generally a single method for applying different kinds of variation operators to members of those population (e.g. limited to sequentially applying to all members of the population one crossover operator and one mutation operator, each with a given probability). However, there are numerous other ways to go, and the strong interaction among all parameters of an Evolutionary Algorithm makes it impossible to a priori decide which way is best.

This paper presents EOlib, a paradigm-free evolutionary computation library, which allows to easily evolve any data structures (objects) that fulfills a small set of conditions. Moreover, algorithms programmed within EOlib are not limited to basic existing EC paradigm like Genetic Algorithms, Evolution strategies, Evolutionary Programming or Genetic Programming, be it at the level of population evolution or variation operator application. Indeed, while all of the above do exist in EO, original experiments can easily be performed using EOlib building blocks.

The rest of the paper is organized as follows: section 2.1 briefly introduces EAs and the basic terminology, and also presents the state of the art in EA libraries. Section 3 presents Evolving Objects, a representation-independent, paradigm-independent, object-oriented approach to Evolutionary Computation. The rest of the paper discusses the EO class library structure in section 4 and surveys some of the existing applications in section 5. Finally, section 6 concludes the paper and presents future lines of work.

2 Introduction

2.1 Evolutionary Algorithms

This section will briefly recall the basic steps of an EA, emphasizing the interdependencies of the different components. The problem at hand is to optimize a given *objective function* over a given search space. A population of individuals (i.e. a P-uple of points of the search space) will undergo some artificial Darwinian evolution, in which the fitness of an individual is directly related to the values the objective function takes at this point.

After a (generally random) intialisation of the population, the generation loop of the algorithm is described in Figure 1

– **Stopping criterion** (and statistics gathering): The simplest stopping criterion is based on the generation counter t (or on the number of function evaluations). However, it is possible to use more complex stopping criteria,

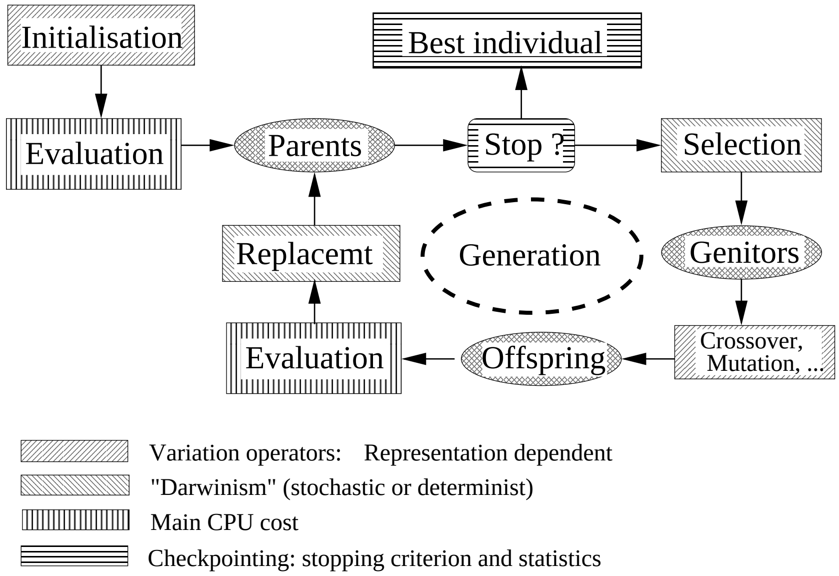

Fig. 1. Sketch of an Evolutionary Algorithm

which depends either on the evolution of the best fitness in the population along generations (i.e., measurements of the gradient of the gains over some number of generations), or on some measure of the diversity of the population.

– **Selection**: Choice of some individuals that will generate offspring. Numerous selection processes can be used, either deterministic or stochastic. All are based on the *fitness* of the individuals, directly related to the objective function. Depending on the selection scheme used, some individuals can be selected more than once. At that point, selected individuals give birth to copies of themselves, the genitors.
– **Application of variation operators**: To each one of these copies some operator(s) are applied, giving birth to one or more offspring. These operators are generally stochastic operators, and one usually distinguish between *crossover* (or *recombination*) and *mutation* operators:
  - crossover operators are operators from E^k (in most cases, $k = 2$) into E, i.e., some parents exchange genetic material to build up one offspring[1].
  - mutation operators are (generally) stochastic operators from E into E.
– **Evaluation**: Computation of the fitnesses of all newborn offspring. As mentioned earlier, the fitness measure of an individual is directly related to its objective function value. Note that in any real-world application, 99% of the total CPU cost of an EA comes from the evaluation part.

[1] Many authors define crossover operators from $E \times E$ into $E \times E$ (two parents generate two offspring), but no significant difference was ever reported between both variants.

- **Replacement**: Choice of which individuals will be part of next generation. The choice can be made either from the set of offspring only (in which case all parents "die") or from both sets of offspring and parents. In either case, the this replacement procedure can be deterministic or stochastic.

The components described above can be categorized into two subsets, that relate to Darwin's principles of natural evolution: the *survival of the fittest* and *small undirected variations.*

- Selection and replacement, also termed *evolution engine*, describe the way Darwinian evolution is applied to the population, The evolution engine is only concerned with the fitness of the individuals, and is totally independent of the representation (the search space).
- Initialisation and variation operators are representation-specific, but have (in most cases) nothing to do with the fitness, following the idea that variations should be undirected.

This basic classification already gives some hints about how to design an evolutionary library, that will be one of the basis of EO design.

2.2 EA libraries

A look at the Genetic Algorithms newsgroup FAQ [19] shows scores of freeware EA libraries; but another look at the GA newsgroups (such as news:comp.ai.genetic also show that very few people actually use them. The rule is home-brew libraries. Most libraries are too hard to use, too restrictive (for instance, restricted to only one EC paradigm), or just plain bad products.

A product stands out among the rest: Matthew's GALib [42], a widely used evolutionary computation library, which includes several paradigms, several representations, and a good deal of variation operators. However, it lacks flexibility in a number of areas.

First, the choice of existing representations is also limited to arrays (of bits, integers or floating point, or any combination), although it can be expanded by sub-classing. However, evolving a neural network, for instance, would mean squeezing it into an array.

Second, it only allows for two variation operators for each genome: mutation and crossover (besides the initialization operator). Moreover, those operators are always applied sequentially, and the only degree of freedom in that respect are the probabilities of application. Hence, for instance, the popular experiment involving an equidistributed random choice of several different mutations is not straightforward. Similarly, there is no simple way to implement Evolution Strategy operators (self-adaptive mutation, or global recombination [2].

Last but not least, only scalar fitness is implemented, which makes it difficult to add constraint handling techniques in a generic way, and almost impossible to do multi-objective optimisation.

3 Evolvable Objects

The library introduced in this paper, EOlib, has flexibility designed from the ground up. This flexibility owns everything to the object-oriented design: every data structure, every operator, every statistic computing routine is an object.

3.1 Data structures

Any data structure can be evolved, if at least one variation operator is provided for such structures. A few pre-existing representations already exist, from the humble bitstring, up to and including GP parse trees and multilayer perceptrons.

What features does a data structure need to be evolvable within EO? It should be **initialisable**; **selectable** and **replicable**; and either **mutable** or **combinable**. These properties will be used as computational analogs for the three criteria for evolution outlined by Maynard-Smith [26], namely heredity, variability and fecundity and will be examined in turn:

- **Initialisability**: This property, while essential in an EA, does not really have a natural counter-part in any of the biological models of evolution (of course, we don't consider here creativism as a model for evolution!). It is generally also given little attention in existing libraries, as standard procedures exist for standard representations. However, even such standard procedures can be questionable in some situations [21]. Whatever, in EO, initialisers are themselves objects, which allows one to use more than one initialisation procedure, a common feature in GP for instance [3].
- **Selectability**: One of the main components of Darwinian evolution is *natural selection*, sometimes also seen as *survival of the fittest*. In EO, like in all EA libraries, all objects are attached a fitness, and that fitness is used to perform such a selection. However, fitnesses in EO are not limited to scalar fitness (see section 3.2 below).
- **Replicability** It should be possible to obtain (possibly imperfect) copies of an object, be it by itself or through the use of other objects (*replicators*). This has a close analogy with the *Criterion of Heredity*. It should also be possible to create objects from scratch, using *object factories*.
- **Mutability** It is the first possible implementation of Maynard-Smith's *Criterion of Variability*, that states that the genotype copying process has imperfections, thus offspring are not equal to the parent(s). Mutation increases the diversity of a population. Mutation operators, or *mutator*, can change an Evolving Object in one or several ways, but the inner workings of the mutation need not be known from outside, neither a particular representation will be needed in order to mutate. The client can only be guaranteed that the object will change in some (generally stochastic) way.
- **Combinability** Another possible variation operator combine two or more objects to create a new one (in a similar way to GA's *crossover*). This is not always possible, but when it is, the operation generally decreases diversity, in the sense that it makes the objects in the population more similar to

each other (although in some cases, such as binary crossover non-respectful of gene boundaries or the Distance Preserving Crossover of Merz and Freisleben [16], it could increase diversity). As it happens with mutation, the exact inner workings of recombination does not need to be known by the client. These objects will usually be called *combinators* or *maters*. One way to ensure a minimal meaningfulness of maters is to follow some of the rules of forma recombination [31]. Since in practice it's generally impossible for combiners to follow all of them [11], each combiner should follow at least one. Combinability can thus serve both as a heredity component and a variability component, this depending on the exact nature of the combination. Balancing heredity and variability is known in the field of Evolutionary Computation as the exploration/exploitation dilemma.

The good news is that most problems solved by computer can be implemented in data structures having these characteristics, including evolutionary algorithms themselves, which have been evolved already by Fogel and coworkers [15], Baeck [1] and Grefenstette [17]; indeed, in the EO framework, algorithms can be just another object, and multilevel evolutionary algorithm can be naturally fitted within the EO framework.

3.2 Fitness function

The fitness in EAs is the only way to specify what represent the natural environment in natural evolution. In most EA libraries, unfortunately, fitness is limited to one single scalar value, and natural selection hence ends up being based on comparisons of those scalar values. However, such choice is very restrictive, and does not make provision for other selection mechanisms, such as selection based on constraints, based on several objectives, or more complex co-evolution processes involving either one population of partial solutions [9] or several competing or cooperating populations [30].

In EOlib, fitness can be of any type (more technically: all Evolutionary Objects are templatized over the fitness type), which opens the door to many other types of EAs. Of course, scalar (real-valued) fitness is still the most widely used, and most popular selections and replacements for real-valued fitness are available. But it is also possible to use fitnesses that are vectors of real numbers, and to design multi-objective [12] or generic constraint-handling selectors [29]. For instance, NSGAII selection [13] and adaptive segregated constraint handling [4] are already implemented in EO.

3.3 Variation operators

Variation operators in EO are objects that exist outside the genotypes they act on: hence any number of variation operators can be designed for the same evolving data structure. Besides, variation operators can take any number of inputs and generate any number of outputs, allowing for instance easy implementation of orgy operators [14] or ES global recombination operators [2]. Furthermore, being separate objects, variation operators can own some private data: for instance, a special selector for choosing the mate of a first partner can be given to a crossover operator, allowing sexual preferences to be taken into account, as in [36, 20]; all these private parameters can then be modified at run-time, allowing easy implementation of e.g. the standard deviation of Gaussian mutations in Evolution Strategies, either following the well-known one-fifth rule [32] or using self-adaptation mechanisms [2].

Variation operators can be combined in different ways. Two basic constructs exist: the sequential combination, in which variation operators are applied one after the other to the whole population with specific rates (as in Simple Genetic Algorithms for instance); the proportional combination, that chooses only one operator among the ones it knows, based on relative pre-defined weights. Furthermore, those two ways of combining variation operators can be recursively embedded. For instance, a very popular combination of operators is to mix different crossovers and different mutations within the Simple GA framework – which amounts to a sequential recombination of a proportional choice among the available crossovers and a proportional choice among the available mutations. Note that these constructs, being themselves objects, can be evolved at run-time, e.g. modifying the different rates based on past evolution.

3.4 Evolution

Evolution engines can be given in different ways: Of course, most popular engines (e.g. Generational GA, Steady-State GA, EP, both ES+ and ES, strategies) are available. But also, all parameters of an evolution engine can be specified in great details: the selection operator and its parameters, the number of offspring to generate, the proportion of strong elitism (best individuals are copied onto the next generation regardless of offspring), the replacement procedure (whether it involves the parents or not), the weak elitism (replace the worst individual in the new population by the best parent if the best fitness is decreasing), ... Hence new evolution engines can be defined simply by assembling existing EO building blocks.

4 Technical description

All the EO ideas have been put in practice in the EOlib class library, an Open Source C++ library which is available from http://eodev.sourceforge.net, together with all facilities of open project in SourceForge: several mailing lists, CVS access to the source tree, bug reporting, ... The current version is 0.9.1, the leading zero in the version indicates that it is not yet complete. EOlib needs an ANSI-C++ compliant compiler, such as the Free Software Foundation gcc (in Linux, other Unix flavors or the CygWin environment for Win95/98/NT); most classes also work with commercial compilers such as Microsoft's Visual C++ 6.0.

Besides the "evolutionary classes" mentioned in the previous section, general facilities for EC applications, such as check-pointing for stopping and restarting applications, multiple statistics gathering, graphic on-line representation that uses gnuplot in Linux are also provided. Moreover, EOlib is open: using existing tutorial template files, implementing one's own new statistics and displaying it on-line, for instance, is straightforward.

There are two ways to use EOlib. The most frequent case is when your representation is already defined in EO (be it bitstrings, real-valued vector or parse-trees), and you simply want to optimize a given fitness function. The only thing that has to be programmed is that fitness computation function, and all other components of the algorithm can simply be input as program parameters.

On the other hand, using an ad hoc representation requires coding the basic representation-dependent parts of the algorithm: the definition of the representation, the initialisation and the variation operators (see section2.1) ... and the fitness function, of course. Similarly, testing a new selection for instance can be done by simply plugging it into an existing EO program, everything else being unchanged. Template files are provided in the tutorial section to help the programmer write his/her own components.

One further plan is to provide an *object repository*, so that if something is programmed using EOlib, the object classes can be immediately posted for everyone to use them. One major outcome would be to improve the reproducibility of EC results: whereas a paper is written using EOlib, the source code of all experiments would be available, and further research could actually use it as a starting point. A link could be added with a paper repository, such as the one the European Evolutionary Computation Network of Excellence, EvoNet, is designing.

5 Applications

EO, so far, has been applied to a number of different areas. The great flexibility of the library has been used to implement complex representations (e.g. multi-layer perceptron, Voronoi diagrams, ...), together with their specific variation operators, multi-objective optimization, specific constraint handling mechanisms, hybrid mutation operators, ...

- *Evolving multilayer perceptrons* [8] no binary or floating point vector representation was used; the objects that were evolved were the multilayer perceptrons themselves. The EO class was used for the population-level operators, but new diversity-generation operators had to be designed: add or eliminate a hidden layer neuron, hidden layer crossover, and mutate initial weights. The back-propagation training algorithm was also used as mutation operator [7]. This application is available also from http://geneura.ugr.es/~pedro/G-Prop.htm.
- *Genetic Mastermind* In the case of the game of Mastermind [27], a GA was programmed to find the hidden combination, improving results ob-

tained in previous implementations.The subject of evolution were the mastermind solutions themselves. The variation operators were also adapted to these objects: a permutation and a *creep* operator, which substituted a number (color) by the next, and the last by the first. A huge improvement was obtained; the algorithm explored only 25% of the space that was explored before [5], that is, around 2% of the total search space, and thus obtained solutions much faster. The game can be played online at http://geneura.ugr.es/~jmerelo/GenMM; the code can be downloaded from the same site.

- Evolution of fuzzy-logic controllers [33]: bidimensional fuzzy-logic controllers were evolved to approximate two-variable functions; variation operators added and subtracted row and columns, and changed values of precedents and consequents. The evolved object approximated the function, and besides, found a proper number of rows and columns for the controller.
- Evolution of RBF neural nets [34]: data structures representing RBFs with diverse radii in each dimension are evolved; variation operators add and subtract RBFs, and change the position of the centers and the value of the radii. Evolved RBFs are usually smaller and more accurate than other found by trial-and-error or incremental procedures.
- Evolutionary voice segmentation [28]: the problem consists in finding the right division of a speech stream, so that different words, phrases, or phonemes can be separated; EO evolves segmentation markers, with very good results. In this case, the evolved data structure are deltas with respect to a linear segmentation.
- As a plug-in to EOlib, a visualization tools that uses Kohonen's Self-Organizing Map [25] has been presented in [35]. This tool presents, after training, a two-dimensional map of fitness to the flattened, one-dimensional vector representation of a chromosome, allowing to assess the evolutionary process by checking that it has explored efficiently the search space.
- A parallel version of EOlib using MPI and PVM is in development; the MPI version has been tested on several benchmark problems [6].
- EOlib has been applied to image segmentation in [39, 40], which applies genetic algorithms to a stripe straightening algorithm used to process and then compress fruit fly embryo images.
- A difficult problem of car engineering, in which the very costly objective function has been replaced by a surrogate cheap model, has been recently tackled using the a combination of multi-objective and constraint-handling techniques (see [37], submitted to the same conference).
- A hybrid surrogate mutation operator has been implemented and tested for parametric optimization. The first results, also submitted to the same conference, are very promising [38].
- Topological optimum design of structures has been a long-time research of on of the authors [24]. However, it was recently ported into EO framework [18] as it is basically a multi-objective problem (minimizing both the weight of the structure and the maximal displacement under a given loading). Within EO, it has been possible to really compare both approaches, as they use

exactly the same representation and variation operators (including the way they are applied).
– *Adaptive Logic Programming* [22] A variable length chromosome was used to steer a path through a logic program in order to generate (constrained) mathematical expressions. Using EO, it was possible to compare the results with a tree-based genetic programming approach [23].

6 Conclusion

In this paper, we had the ambitious objective of presenting a new framework for evolutionary computation called EO, that would include all evolutionary computation paradigms as well as new ones, with novel data structures evolved, general or particular variation operators, and any population-level change operators.

EO has a practical implementation in the shape of the EO class library, which is public and freely available under the LGPL (FSF's Library, or lesser, general public license) from http://eodev.sourceforge.net. This library has already been applied to problems in which, traditionally, binary or floating point representations were used, using instead as evolving object the same data structure one want to obtains as a result, such as a neural net or a bidimensional fuzzy logic controller.

As possible lines of future work, we will try the implementation of EOlib in different OO languages, such as Java, and its interoperability with each other. Another feature is an application generator, that will use high-level evolutionary computation languages such as EASEA [10], and an operating-system independent graphical user interface.

Acknowledgements

This work has been supported in part by FEDER I+D project 1FD97-0439-TEL1, CICYT TIC99-0550, and INTAS 97-30950.

References

1. T. Bäck. Self-adaptation in genetic algorithms. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life. Toward a Practice of Autonomous Systems*, pages 263–271, MIT Press, Cambridge, MA.
2. Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
3. W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.
4. S. BenHamida and M. Schoenauer. An adaptive algorithm for constrained optimization problems. In M. Schoenauer et al., editor, *Proceedings of the* 6^{th} *Conference on Parallel Problems Solving from Nature*, pages 529–539. Springer-Verlag, LNCS 1917, 2000.

5. J. L. Bernier, C. Ilia Herráiz, J. J. Merelo, S. Olmeda, and A. Prieto. Solving *mastermind* using GAs and simulated annealing: a case of dynamic constraint optimization. In *Parallel Problem Solving from Nature IV*, pages 554–563. Springer-Verlag, LNCS 1141, 1996.
6. J. G. Castellano, M. García-Arenas, P. A. Castillo, J. Carpio, M. Cillero, J. J. Merelo, A. Prieto, V. Rivas and G. Romero. Objetos evolutivos paralelos. In *XI Jornadas de Paralelismo*, Universidad de Granada Depto. ATC, pages 247–252, 2000.
7. P. A. Castillo, J. González, J. J. Merelo, A. Prieto, V. Rivas, and G. Romero. G-Prop-III: Global optimization of multilayer perceptrons using an evolutionary algorithm. In *GECCO99*, 1999.
8. P.A. Castillo, J.J. Merelo, V. Rivas, G. Romero, and A. Prieto. Evolving Multilayer Perceptrons. *Neural Processing Letters* 12(2):115–127, 2000.
9. P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Polar ifs + individual gp = efficient inverse ifs problem solving. *Genetic Programming and Evolvable Machines*, 1(4), 2000.
10. P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it easea. In M. Schoenauer et al., editor, *Proceedings of the* 6^{th} *Conference on Parallel Problems Solving from Nature*, pages 891–901. Springer Verlag, LNCS 1917, 2000.
11. Carlos Cotta, Enrique Alba, and José M. Troya. Utilizing dynastically optimal forma recombination in hybrid genetic algorithms. In Thomas Back Agoston E. Eiben, Marc Schoenauer, editor, *Parallel Problem Solving From Nature – PPSN V*, pages 305-314. Springer Verlag, LNCS 1498, 1998.
12. K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Chichester, UK: Wiley, 2001.
13. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In M. Schoenauer et al., editor, *Proceedings of the* 6^{th} *Conference on Parallel Problems Solving from Nature*, pages 849–858. Springer-Verlag, LNCS 1917, 2000.
14. A.E. Eiben, P.-E. Raue, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Proceedings of the* 3^{rd} *Conference on Parallel Problems Solving from Nature*, pages 78–87. Springer Verlag, LNCS 866, 1994.
15. D. B. Fogel, L. J. Fogel, and J. W. Atmar. Meta-evolutionary programming. In R. R. Chen, editor, *Proceedings of 25th Asilomar Conference on Signals, Systems and Computers*, pages 540–545, Pacific Grove, California, 1991.
16. B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Oriceedubgs if tge 1996 IEEE International Conference on Evolutionary Computation*, pages 616–621. IEEE Press, 1996.
17. J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics, SMC-16*, 1986.
18. F. Jouve H. Hamda, E. Lutton, M. Schoenauer, and M. Sebag. Compact unstructured representations in evolutionary topological optimum design. *Intl J. of Applied Intelligence*, 2001. To appear.
19. Jōrg Heitkōter and David Beasley. The hitch-hiker's guide to evolutionary computation, (FAQ for `comp.ai.genetic`). Available from `http://surf.de.uu.net/encore/www/`.
20. R. Hinterding and Z. Michalewicz. Your brain and my beauty. In D.B. Fogel, editor, *Proceedings of the Fifth IEEE International Conference on Evolutionary Computation*, IEEE Press, 1998.

21. L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In Th. Bäck, editor, *Proceedings of the 7^{th} International Conference on Genetic Algorithms*, pages 268–275. Morgan Kaufmann, 1997.
22. M. Keijzer, V. Babovic, C. Ryan, M. O'Neill and M. Cattolico Adaptive Logic Programming. In *GECCO01*, 2001.
23. M. Keijzer, C. Ryan, M. O'Neill, M. Cattolico and V. Babovic Ripple Crossover in Genetic Programming. In *EuroGP 2001*, 2001.
24. C. Kane and M. Schoenauer. Topological optimum design using genetic algorithms. *Control and Cybernetics*, 25(5):1059–1088, 1996.
25. Teuvo Kohonen. *Self-Organizing Maps.* Springer, Berlin, Heidelberg, 1995.
26. J. Maynard-Smith. *The theory of evolution.* Penguin, 1975.
27. J. J. Merelo, J. Carpio, P. Castillo, V. M. Rivas, and G. Romero. Finding a needel in a haystack using hints and evolutionary computation: the case of genetic mastermind. In *Late breaking papers at the GECCO99*, pages 184–192, 1999.
28. J. J. Merelo and D. Milone. Evolutionary algorithm for speech segmentation. *Submitted*, 2001.
29. Z. Michalewicz and M. Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.
30. J. Paredis. Coevolutionary computation. *Artificial Life*, 2:355–375, 1995.
31. N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–20, 1991.
32. I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution.* Fromman-Hozlboog Verlag, Stuttgart, 1973.
33. V.M. Rivas, J. J. Merelo, I. Rojas, G. Romero, P.A. Castillo, and J. Carpio. Evolving 2-dimensional fuzzy logic controllers. Submitted.
34. V. Rivas, P. Castillo, and J. J. Merelo. Evolving RBF neural nets. In *Proceedings IWANN'2001*, Springer-Verlag, LNCS, 2001. To appear.
35. G. Romero, M. García-Arenas, J. G. Castellano, P. A. Castillo, J. Carpio, J. J. Merelo, A. Prieto, and V. Rivas. Evolutionary computation visualization: Application to G-PROP. pages 902–912. Springer, LNCS 1917, 2000.
36. E. Ronald. When selection meets seduction. In L. J. Eshelman, editor, *Proceedings of the 6^{th} International Conference on Genetic Algorithms*, pages 167–173. Morgan Kaufmann, 1995.
37. O. Roudenko, T. Bosio, R. Fontana, and M. Schoenauer. Optmization of car front crash members. In *EA'01*, 2001. Submitted.
38. K. Abboud, and M. Schoenauer. Hybrid surrogate mutation: preliminary results. In *EA'01*, 2001. Submitted.
39. A.V. Spirov, D.L. Timakin, J. Reinitz, and D Kosman. Experimental determination of drosophila embryonic coordinates by genetic algorithms, the simplex method, and their hybrid. In *Proceedings of Second European Workshop On Evolutionary Computation In Image Analysis And Signal Processing*, April 2000.
40. A.V. Spirov and J. Reinitz. Using of genetic algorithms in image processing for quantitative atlas of drosophila genes expression. Available from `http://www.mssm.edu/molbio/hoxpro/atlas/atlas.html` .
41. A. Tang. Constructing GA applications using TOLKIEN. Technical report, Dept. Computer Science, Chinese University of Hong Kong, 1994.
42. M. Wall. Overview of GALib. `http://lancet.mit.edu/ga`, 1995.