Chapter 4
A Primer on Memetic Algorithms

Ferrante Neri and Carlos Cotta

4.1 Introduction

Memetic Algorithms (MAs) are population-based metaheuristics composed of an evolutionary framework and a set of local search algorithms which are activated within the generation cycle of the external framework, see [376]. The earliest MA implementation has been given in [621] in the context of the Travelling Salesman Problem (TSP) while an early systematic definition has been presented in [615]. The concept of meme is borrowed from philosophy and is intended as the unit of cultural transmission. In other words, complex ideas can be decomposed into memes which propagate and mutate within a population. Culture, in this way, constantly undergoes evolution and tends towards progressive improvements. Strong ideas tend to resist and be propagated within a community while weak ideas are not selected and tend to disappear. In the metaphor, the ideas are the search operators: the fittest tend to be employed while the inadequate ones are likely to disappear.

This chapter gives an initial description of MA frameworks explaining the literature context of their generation and success as well as their general structures. More specifically, Section 4.2 analyzes the context where MAs have been introduced and puts into relationship the algorithmic flexibility of the memetic paradigm with the the No Free Lunch Theorem. Section 4.3 shows the outline of a general MA implementation. Section 4.5 gives a quick overview on the MA application and employment in literature. Finally, Section 4.6 explains the difference between MAs and the general emerging trend of Memetic Computing.

Ferrante Neri
Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014, University of Jyväskylä, Finland
e-mail: ferrante.neri@jyu.fi

Carlos Cotta
Departamento de Lenguajes y Ciencias de la Computación, Escuela Técnica Superior de Ingeniería Informática, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain
e-mail: ccottap@lcc.uma.es

F. Neri et al. (Eds.): Handbook of Memetic Algorithms, SCI 379, pp. 43–52.
springerlink.com

4.2 The Need for Memetic Algorithms

In order to understand in depth the role and need of MAs, it is fundamental to consider the historical context within which MAs have been defined. In 1988, when the first MAs were defined, Genetic Algorithms (GAs) were extremely popular among computer scientists and their related research was oriented towards the design of algorithms having a superior performance with respect to all the other algorithms present in literature. This approach is visible in many famous texts published in those years, e.g. [325]. Unlike all the algorithms proposed at that time, a MA was not a specific algorithm but was something much more general than an optimization algorithm: since MAs consist of the concept of combining global and local search algorithms, they represented a broad and flexible class of algorithms which somehow contained the previous work on Evolutionary Algorithms (EAs) and thus, constituted a new philosophy in optimization. Probably due to their excessively innovative contents, MAs had to face for about one decade, the skepticism of the scientific community which repeatedly rejected the memetic approach as a valuable possibility in optimization.

Since 1997, researchers in optimization had to dramatically change their view about the subject. More specifically, in the light of increasing interest in general purpose optimization algorithms, it has become important, in the end of 90's to understand the relationship between how well an algorithm a performs on a given optimization problem f on which it is run on the the basis of the features of the problem f. A slightly counter intuitive result has been derived by Wolpert and Macready in [940] which states that for a given pair of algorithms A and B:

$$\sum_{f} P(x_m|f,A) = \sum_{f} P(x_m|f,B) \tag{4.1}$$

where $P(x_m|f,A)$ is the probability that algorithm A detects the optimal solution for a generic objective function f and $P(x_m|f,B)$ is the analogue probability for algorithm B. In [940] the statement eq. 4.1 is proved for both static and time-dependent case and are named "No Free Lunch Theorems" (NFLT). In other words, in 1997 it was mathematically proved that the average performance of any pair of algorithms across all possible problems is identical. Thus, if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems as this is the only way that all algorithms can have the same performance averaged over all functions [940]. Strictly speaking, the proof of NFLT is made under the hypothesis that both the algorithms A and B are non-revisiting, i.e. the algorithms do not perform the fitness evaluation of the same candidate solution more often than once during the optimization run. Although this hypothesis is de facto not respected for most of the computational intelligence optimization algorithms, the concept that there is no universal optimizer had a significant impact on the scientific community.

It should be highlighted that a class of problems on which an algorithm performs well is not defined by the nature of the application but rather by the features of the

fitness function within the search space. For example an optimization problem is characterized by:

- the shape and properties of a corresponding fitness landscape (see definitions below),
- multi-modality,
- separability of the problem,
- absence or presence of a noise in the values of the objective function (optionally, the type of noise),
- time dependency of the objective function (dynamic problems)
- shape and connectivity of the search domain

In evolutionary biology, the idea of studying evolution by visualizing the distribution of fitness values as a kind of landscape was first introduced by Wright [941].

More formally, the fitness landscape (S, f, d) of a problem instance for a given problem consists of a set of points S, a fitness function f which assigns values (fitness) to solutions from S, and a distance measure $d : S \times S \rightarrow \mathbb{R}$ which defines the spacial structure of the landscape. This rather abstract concept has proven to be useful for understanding the functionality of various optimization methods, see [581] and [583].

One of the most important properties of the fitness landscape is epistasis whose concept has been borrowed from biology where it refers to the degree to which the genes are correlated. As it is well known, a function is separable if it can be rewritten as a sum of functions of just one variable. The separability is closely related to the concept of epistasis. In the field of evolutionary computation, the epistasis measures how much the contribution of a gene to the fitness of the individual depends on the values of other genes. Nonseparable functions are more difficult to optimize as the accurate search direction depends on two or more genes. On the other hand, separable functions can be optimized for each variable in turn. However, epistasis does not provide any piece of information on how the fitness values are topologically related to each other. By knowing the epistasis of an optimization problem, it cannot be established whether the fitness values form a smooth progression resulting in a solitary optimum or whether they form a spiky pattern of many isolated optima [438].

The impossibility of understanding each detail of the fitness landscape depends not only on the fitness function but also on the search algorithm [438] since an observed landscape appears to be an artefact of the algorithm used or, more specifically, of the neighborhood structure induced by the operators used by the algorithm [433]. The neighborhood structure is defined as a set of points that can be reached by a single move of a search algorithm [375]. Closely related to the concept of the neighborhood structure is the notion of a basin of attraction induced by this structure. More specifically, a basin of attraction of a local optimum x is the set of points X of the search space such that a search algorithm starting from any point from X ends in the local optimum x. A special note should be made regarding the landscapes with plateaus, i.e. regions in search domain where the function has constant or nearly constant values. If a search method is trapped on such region it cannot get

any information regarding the gradient or even its estimates. Generally speaking, this situation is rather complicated and special algorithmic components should be used in this case. Finally, an important feature of a fitness landscape is the presence or absence of symmetry. Special components can be included in the algorithms for symmetrical problems.

In addition, two features can be mentioned which appear to be semi-defining when distinguishing the classes of problems on which an algorithm performs well. The first one is dimensionality of the problem. Two problems with high dimensionality of the search domain can be put into the same class, however an algorithm that performs well for one of them might not necessarily work well for the other one. At the same time, two specialized algorithms for these two problems will have some common features intended to overcome difficulties arising from high dimensionality. The second semi-defining feature is computational cost of a single evaluation of the objective function. Clearly, two problems with computationally expensive objective functions can have different features mentioned above that will put them into different classes. However, these problems are unsolvable (in practice) if treated as computationally cheap functions, therefore algorithms for such problems should have common type components which allow proper handling of the computational cost.

There is generally a performance advantage in incorporating prior knowledge into the algorithm, however the results of NFLT do not deem the use of unspecialized algorithms futile. It is impossible to determine the fraction of practical problems for which an algorithm yields good results rapidly, therefore a practical free lunch is possible. NFLT constitute, in a certain sense, the "Full Employment Theorem" (FET) for optimization professionals. In computer science and mathematics, the term FET is used to refer to a theorem that shows that no algorithm can optimally perform a particular task done by some class of professionals. In this sense, as no efficient general purpose solver exists, there is always scope for improving algorithms for better performance on particular problems. Since MAs, as mentioned above, represent a broad class of algorithms which combine various algorithmic components, a suitable combination is necessary for a given problem. Since, during the last decade, computer scientists had to observe the features of their optimization problem in order to propose an ad-hoc optimization algorithm, the approach of combining various search operators within the algorithmic design became a common practice. In this sense, the development of NFLT implicitly encouraged the use and development of MAs, which became extremely popular and often necessary, in computer science at first, and in engineering and applied science more recently, thus constituting the FET for MAs.

4.3 A Basic Memetic Algorithm Template

As mentioned in previous sections, MAs blend together ideas from different search methodologies, and most prominently ideas from local search techniques and population-based search. Indeed, from a very general point of view a basic MA

can be regarded as one (or several) local search procedure(s) acting on a set pop of $|pop| \geqslant 2$ solutions which engage in periodical episodes of cooperation via recombination procedures. This is shown in Algorithm 4.

Algorithm 4. A Basic Memetic Algorithm

```
1  function BasicMA (in P: Problem, in par: Parameters): Solution;
2  begin
3      pop ← Initialize(par, P);
4      repeat
5          newpop_1 ← Cooperate(pop, par, P);
6          newpop_2 ← Improve(newpop_1, par, P);
7          pop ← Compete (pop, newpop_2);
8          if Converged(pop) then
9              pop ← Restart(pop, par);
10         endif
11     until TerminationCriterion(par) ;
12     return GetNthBest(pop, 1);
13 end
```

This template requires some explanation. First of all, the Initialize procedure is responsible for creating the initial set of $|pop|$ solutions. While traditional evolutionary algorithms usually resorted to simply generating $|pop|$ solutions at random (in some cases following a systematic procedure to ensure a good coverage of the search space), MAs typically attempt to use high-quality solutions as starting point. This can be done either using a more sophisticated mechanism (for instance, some constructive heuristic) to inject good solutions in the initial population [861], or by using a local-search procedure to improve random solutions (see Algorithm 5).

Algorithm 5. Injecting high-quality solutions in the initial population.

```
1  function Initialize(in par: Parameters, in P: Problem): Bag{Solution};
2  begin
3      pop ← ∅;
4      for j ← 1 to par.popsize do
5          i ← RandomSolution(P);
6          i ← LocalSearch (i, par, P);
7          pop ← pop ∪ {i};
8      endfor
9      return pop;
10 end
```

As for the TerminationCriterion function, it typically amounts to checking a limit on the total number of iterations, reaching a maximum number of iterations without improvement, or having performed a certain number of population restarts.

Algorithm 6. The pipelined Cooperate procedure.

```
1  function Cooperate (in pop: Bag{Solution}, in par: Parameters, in P: Problem):
   Bag{Solution};
2  begin
3      lastpop ← pop;
4      for j ← 1 to par.numop do
5          newpop ← ∅;
6          for k ← 1 to par.numapps^j do
7              parents ← Select (lastpop, par.arity^j);
8              newpop ← newpop ∪ ApplyOperator (par.op^j, parents, P);
9          endfor
10         lastpop ← newpop;
11     endfor
12     return newpop;
13 end
```

The procedures Cooperate and Improve constitute the core of the MA. Starting with the former, its most typical incarnation is based on two operators for selecting solutions from the population and recombining them. Of course, this procedure can be readily extended to use a collection of variation operators applied in a pipeline fashion. As shown in Algorithm 6, this procedure comprises *numop* stages, each one corresponding to the iterated application of a particular operator op^j that takes $arityin^j$ solutions from the previous stage, generating $arityout^j$ new solutions.

As to the Improve procedure, it embodies the application of a local search procedure to solutions in the population. Notice that in an abstract sense a local search method can be modeled as a unary operator, and hence it could have been included within the Cooperate procedure above. However, local search plays such an important role in MAs that it deserves separate treatment. Indeed, there are several important design decisions involved in the application of local search to solutions, i.e., to which solutions should it be applied, how often, for how long, etc. See also next section.

Next, the Compete procedure is used to reconstruct the current population using the old population *pop* and the newly generated population $newpop_2$. Borrowing the terminology from the evolution strategy [761, 800] community, there exist two main possibilities to carry on this reconstruction: the *plus* strategy and the *comma* strategy. The latter is usually regarded as less prone to stagnation [32], with the ratio $|newpop|/|pop| \simeq 6$ being a common choice [34]. Since this option can be somewhat computationally expensive if the fitness function is complex and time-consuming, a popular alternative is using a plus strategy with a low value of $|newpop|$, analogous to the so-called *steady-state* replacement strategy in GAs [930]. This option usually provides a faster convergence to high-quality solutions, although care has to be taken with premature convergence to suboptimal regions of the search space. This leads to the last component of the template shown in Algorithm 4, the restarting procedure.

Algorithm 7. The Restart procedure.

```
1  function Restart (in pop: Bag{Solution}, in par: Parameters, in P: Problem):
   Bag{Solution};
2  begin
3      newpop ← ∅;
4      for j ← 1 to par.preserved do
5          i ← GetNthBest(pop, j);
6          newpop ← {i};
7      endfor
8      for j ← par.preserved + 1 to par.popsize do
9          i ← RandomSolution(P);
10         i ← LocalSearch (i, par, P);
11         newpop ← {i};
12     endfor
13     return newpop;
14 end
```

First of all, it must be decided whether the population has degraded or has not, using some measure of information diversity in the population such as Shannon's entropy [184]. Once the population is considered to be at a degenerate state, the restart procedure is invoked. Again, this can be implemented in a number of ways. A very typical strategy is to keep a fraction of the current population, generating new (random or heuristic) solutions to complete the population, as shown in Algorithm 7. The procedure shown therein is also known as the *random-immigrant* strategy [130]. Another possibility is to activate a *strong* or *heavy* mutation operator in order to drive the population away from its current location in the search space.

4.4 Design Issues

The general template of MAs depicted in the previous section must be instantiated with precise components in order to be used for solving a specific problem. MAs are commonly implemented as EAs endowed with a local search component, and therefore the theoretical corpus available for the former can be used to guide some aspects of the design process, e.g., the representation of solutions in terms of meaningful information units [183, 751].

The most MA-specific design decisions are those related to the local search component, not just from the point of view of parameterization (see below) but also with the actual inner working of the component and its interplay with the remaining operators. This latter issue is well exemplified in the work of Merz and Freisleben on the TSP [285]. They consider the use of the Lin-Kernighan heuristic [524], a highly intensive local search procedure, and note that the average distance between local optima is similar to the average distance between a local optimum and the global optimum. For this reason, they introduce a distance-preserving crossover (DPX) operator that generate offspring whose distance from the parents is the same as the

distance between the parents themselves. Such an operator is likely to be less effective if a less powerful local improvement method, e.g., 2-opt, was used, inducing a different distribution of local optima.

Once a local search procedure is selected, an adequate parameterization must be determined, i.e., how often it must be applied, how to select the solutions that will undergo local improvement, and how long must improvement epochs last. These are delicate issues since there exists theoretical evidence [494, 857] that an inadequate parameter setting can turn the algorithmic solution from easily solvable to non-polynomially solvable. Regarding the probability of application of local search, its precise values largely depend on the problem under consideration [411], and its determination is in many cases an art. For this reason, adaptive and self-adaptive mechanisms have been defined in order to let the algorithm learn what the most appropriate setting is. The term partial lamarckianism [151, 396, 717] is used to denote these strategies where not every individual is subject to local search.

As to the selection of individuals that will undergo local search, most common options are random-selection, and fitness-based selection, where only the best individuals are subject to local improvement. For example, Nguyen *et al.* [665] consider an approach in which the population is sorted and divided into n levels (n being the number of local search applications), and one individual per level is randomly selected. Note that such a strategy can be readily deployed on a structured MA as defined by Moscato *et al.* [62, 94, 282, 576, 578], in which fitness-based layers are explicitly available. See also [80, 736, 737, 836] for other population management strategies.

4.5 Conclusions and Outlook

Memetic algorithms are a pragmatic, cross-disciplinary optimization paradigm that has emerged in the last quarter of a century to become nowadays one of the most widely used solving approaches. This is supported by a plethora of applications in disparate fields ranging from machine learning and knowledge discovery to planning, scheduling and timetabling, from bioinformatics to electronics, engineering, and telecommunications, or from economics to physics, just to mention a few. The reader may check [154, 375, 618, 619, 620, 626, 632], for a survey of these applications and pointers to the literature.

Throughout this chapter we have provided a brief introduction to the main issues regarding the definition and design of a basic memetic algorithm. However, it must be emphasized that the MA paradigm is very rich and has given rise to an ample set of variations and more sophisticated MA models. Among these, we can firstly cite multiobjective MAs (MOMAs). MOMAs are applied to problems which exhibit multiple, partially-conflicting objectives, and in which the notion of Pareto-dominance is therefore essential. Actually, MOMA approaches can be roughly classified into two major classes: scalarizing approaches [408, 409, 419, 421] (based on the use of some aggregation mechanism to combine the multiple objectives into a single scalar value), and Pareto-based approaches [471, 472] (considering the

notion of Pareto-dominance for deciding transitions among neighboring solutions). MOMAs will be dealt in more detail in chapter 13 in this volume.

Adaptive MAs also deserve special attention. As mentioned in Section 4.4, decisions related to parameterization are essential in order to achieve an effective MA. It is therefore not surprising that attempts have been made to let the algorithm find by itself adequate values for these parameters [40, 536, 605, 606]. Furthermore, the term "meta-lamarckian learning" [680] has been coined to denote strategies in which the algorithm learns to select appropriate local search operators from a certain available collection (note the relationship with hyperheuristics [169]). A further step is taken in the so-called multi-memetic algorithms, in which each solution carries a gene that indicates which local search has to be applied on it (either indicating which one from a pre-existing collection, by parameterizing a general local search template, or by using a grammar to define new operators) [488, 490, 496]. At an even higher level, solutions and local-search operators can coevolve [830, 831]. Adaptive MAs will be dealt in more detail in chapter 11 in this volume.

Last but not least, there exist nowadays a growing trend in combining MAs with complete techniques such as branch-and-bound or branch-and-cut among others. There are many ways in which such a combination can be done. For example, an exact technique can be used as an internal operator of the MA [295, 742], as a post-processing technique [469], run in parallel with the MA [294, 297, 740], and even combine several of the previous approaches [299]. The combination of MAs with exact techniques will be dealt in more detail in chapter 12 in this volume.

4.6 Memetic Algorithms and Memetic Computing

It is fundamental to clarify the difference between MAs and Memetic Computing (MC) . As stated above, MAs are population-based evolutionary algorithms composed of an evolutionary framework and a list of local search algorithms activated within the generation cycle of the evolutionary framework, see [376]. While this book refers to MAs, it is worthy to take into account that recently the term MC became widely used amongst computer scientists. An early definition has been given in [689], where MC is defined as "...a paradigm that uses the notion of meme(s) as units of information encoded in computational representations for the purpose of problem solving". In other words, part of the scientific community tried to extend the concept of meme for problem solving, see [655], to something broader and more innovative. The fact that ad-hoc optimization algorithms can efficiently solve given problems is a well-known result from literature. On the other hand, the ultimate goal in artificial intelligence is the generation of autonomous and intelligent structures. In computational intelligence optimization, the goal is the automatic detection of the optimal optimization algorithm for each fitness landscape, or, in other terms, the on-line (i.e. during run-time) automatic design of optimization algorithms. MC can be seen then as a subject which studies complex structures composed of simple modules (memes) which interact and evolve adapting to the problem in order to solve it. This view of the subject leads to a more modern definition of MC.

Definition 4.1. Memetic Computing is a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems.

In this light, MAs should be seen as a cornerstone and founding subset of MC.

Acknowledgements. C. Cotta is supported by Spanish MICINN under project NEMESIS (TIN2008-05941) and by Junta de Andalucía under project TIC-6083. F. Neri is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.