Memetic Evolutionary Algorithms

W.E. Hart[1], N. Krasnogor[2], and J.E. Smith[3]

[1] Sandia National Laboratory
Albuquerque, New Mexico
USA
[2] Automatic Scheduling, Optimisation and Planning Group
School of Computer Science and IT
University of Nottingham, U.K.
http:\\www.cs.nott.ac.uk\~nxk
natalio.krasnogor@nottingham.ac.uk
[3] Faculty of Computing, Engineering and Mathematical Sciences,
University of the West of England,
Bristol BS16 12QY, U.K.
james.smith@uwe.ac.uk
http://www.cems.uwe.ac.uk/ jsmith

1 Summary

Memetic Evolutionary Algorithms (MAs) are a class of stochastic heuristics for global optimization which combine the parallel global search nature of Evolutionary Algorithms with Local Search to improve individual solutions. These techniques are being applied to an increasing range of application domains with successful results, and the aim of this book is both to highlight some of these applications, and to shed light on some of the design issues and considerations necessary to a successful implementation. In this chapter we provide a background for the rest of the volume by introducing Evolutionary Algorithms (EAs) and Local Search. We then move on to describe the synergy that arises when these two are combined in Memetic Algorithms, and to discuss some of the most salient design issues for a successful implementation. We conclude by describing various other ways in which EAs and MAs can be hybridized with domain-specific knowledge and other search techniques.

2 Introduction

Memetic Algorithms (MAs) are a class of stochastic global search heuristics in which Evolutionary Algorithms-based approaches are combined with local search techniques to improve the quality of the solutions created by evolution. MAs have proven very successful across a wide range of problem domains such

as combinatorial optimization [27], optimization of non-stationary functions [42], and multi-objective optimization [20] (see [29] for an extensive bibliography).

Methods for hybridizing EAs with local search have been given various names in research papers such as: *hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms*, and others. Moscato [3] coined the name *memetic algorithm* to cover a wide range of techniques where evolutionary-based search is augmented by the addition of one or more phases of local search.

The natural analogies between human evolution and learning, and EAs and artificial neural networks (ANNs) prompted a great deal of research into the use of MAs to evolve the structure of ANNs. ANNs were trained using back-propagation or similar means during the 1980s and early 1990s. However, research applying MAs to ANNs gave a great deal of insight into the role of learning, Lamarckianism, and the Baldwin effect to guide evolution (e.g. [8, 7, 8, 9, 10, 11, 12] amongst many others). This research reinforced the experience of "real-world" practitioners as to the usefulness of incorporating local search and domain-based heuristics within an EA framework.

Since then a number of PhD theses [14, 25, 15, 27, 16] have developed algorithmic analyses of MAs. These analyses and related empirical results demonstrate the potential impact of MAs, and in practice, many state-of-the-art EAs employ some element of hybridization using local search. Research in MAs is now sufficiently mature and distinct to have its own annual international workshop, and an extensive on-line bibliography of MA research is maintained at [29].

In this chapter we set the scene for the rest of this book by providing brief introductions to Evolutionary Algorithms (EAs) and Local Search (LS). We also discuss some of the issues which arise when hybridizing the two to create MAs. As our aim is to provide an overview, we cannot hope to give a detailed description of either EAs or the many LS methods available. There are wide variety of books discussing these methods that the user can read for further detail (e.g., see [17, 18]). The rest of this chapter is organized as follows:

- In Section 3 we provide a brief overview and historical background to the field of Evolutionary Algorithms, focusing particularly on their use as search and optimization techniques.
- In Section 4 we provide a brief introduction to local search and some related techniques.
- In Section 5 we discuss some of the motives and rationale underpinning the hybridization of EAs with other search technologies and motivate this book's focus on Memetic Algorithms. Our focus is on EA hybrids in which LS acts on the output of evolutionary operators, that is to say in which some form of "lifetime learning" or "plasticity" is incorporated into the "standard" evolutionary cycle..

- In Section 6 we discuss some of the design issues that must be considered when implementing an MA.
- Finally, in Section 7 we discuss the *structure* of evolutionary and memetic algorithms, and consider various places within the evolutionary cycle that other heuristics and or domain specific knowledge may be incorporated.

3 A Brief Introduction to Evolutionary Algorithms

The idea of applying Darwinian principles to automated problem solving dates back to the forties, long before the breakthrough of computers [19]. As early as 1948, Turing proposed "genetical or evolutionary search", and by 1962 Bremermann had actually executed computer experiments on "optimization through evolution and recombination". During the 1960s three different implementations of the basic idea were developed in different places. In the USA, Fogel, Owens, and Walsh introduced **evolutionary programming** [20, 21], while Holland called his method a **genetic algorithm** [22, 23, 24]. Meanwhile, in Germany, Rechenberg and Schwefel invented **evolution strategies** [25, 26]. For about 15 years these areas developed separately; but since the early 1990s they have been viewed as different representatives of a common technology that has come to be known as **evolutionary computing** (EC) [27, 28, 29, 30, 31]. In the early 1990s a fourth methodology following the same general ideas emerged, **genetic programming**, championed by Koza [32, 33, 34]. The contemporary terminology denotes the whole field by evolutionary computing, and the algorithms involved are termed **evolutionary algorithms**; evolutionary programming, evolution strategies, genetic algorithms, and genetic programming are subareas belonging to the corresponding algorithmic variants.

3.1 The Principal Metaphor

The common underlying idea behind different evolutionary algorithms is the same: given a population of individuals, mechanisms adapted from natural selection and genetic variation are used to evolve individuals with high fitness. Given a quality function to be maximized, we can randomly create a set of candidate solutions, i.e., elements of the function's domain, and apply the quality function as an abstract fitness measure – the higher the better. Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination and/or mutation to them. Recombination is an operator applied to two or more selected candidates (the so-called parents) and results in one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Executing recombination and mutation leads to a set of new candidates (the offspring) that compete – based on their fitness (and possibly age)– with the old ones for a place in the next generation. This process can be iterated until a candidate

with sufficient quality (a solution) is found or a previously set computational limit is reached.

In this process there are two fundamental forces that form the basis of evolutionary systems:

- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty.

- Selection filters, and induces constraints on, candidate solutions.

The combined application of variation and selection generally leads to improving fitness values in consecutive populations. It is easy to view such an evolutionary process as optimizing by iteratively generating solutions with increasingly better values. Alternatively, evolution it is often seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements. Matching these requirements more closely implies an increased viability, reflected in a higher number of offspring. The evolutionary process makes the population increasingly better at being adapted to the environment.

The general scheme of an evolutionary algorithm is shown in Figure 1 in a pseudocode fashion. It is important to note that many components of evolutionary algorithms are stochastic. During selection, fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive. For recombination of individuals the choice of which pieces will be recombined is random. Similarly for mutation, the pieces that will be mutated within a candidate solution, and the new pieces replacing them, are chosen randomly.

```
:
Begin
  INITIALIZE population with random candidate solutions;
  EVALUATE each candidate;
  Repeat Until ( TERMINATION CONDITION is satisfied ) Do
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  endDo
End.
```

Fig. 1. The general scheme of an evolutionary algorithm in pseudocode.

3.2 Components of an EA

Representation

Solutions to the problem being solved are usually referred to as **phenotypes**. Phenotypes are indirectly manipulated by the EA variation and selection operators by virtue of being encoded in **genotypes**. Genotypes within the EA population are the objects upon which the operators act.

The **representation** employed by an EA can thus be represented by a ternary relation $R = (\mathcal{P}, \mathcal{G}, \mathcal{M})$ that specifies the relationship between the space of phenotypes,$\mathcal{P}$, and the space of genotypes $\mathcal{G}$. The mapping $\mathcal{M}$ is a function with domain in $\mathcal{G}$ and range in $\mathcal{P}$ that provides the "interpretation" of the representation. For example, a phenotypic space of real values, $\mathcal{P} \equiv \Re$, can be easily encoded by a binary genotypic search space, $\mathcal{G} \equiv \{0,1\}^+$, using as a mapping $\mathcal{M}$ a gray coding. That is, the gray coding defines *how* binary strings are to be mapped to, or interpreted into, real values.

It is important to understand that the phenotype space can be very different from the genotype space, and thus the EA designer must ensure that the (optimal) solution to the problem at hand – a phenotype – can be represented in the given genotype space.

The common EC terminology uses many synonyms for naming the elements of these two spaces. On the side of the original problem context, **candidate solution**, phenotype, and **individual** are used to denote points of the space of possible solutions. This space itself is commonly called the **phenotype space**. On the side of the EA, genotype, **chromosome**, and again individual can be used for points in the space where the evolutionary search actually takes place. This space is often termed the **genotype space**. There are also many synonymous terms for the elements of individuals. A placeholder is commonly called a variable, a **locus** (plural: loci), a position, or – in a biology-oriented terminology – a **gene**. An object on such a place can be called a value or an **allele**.

Evaluation Function

The **evaluation function** represents the quality of an individual. It forms the basis for selection, and thereby it facilitates improvements. More accurately, it defines what improvement means. From the problem-solving perspective, it provides the measure with which alternative solutions can be compared. The evaluation function is commonly called the **fitness function** in EC. Problems typically solved by EAs are optimization problems, which are specified with an **objective function**. For minimization problems, an evaluation function is commonly formed by negating the objective function.

Population

A **population** is a set of possible solutions. Specifically, a population is a multiset of genotypes.[4] In some sophisticated EAs, a population has an additional spatial structure, with a distance measure or a neighborhood relation. In such cases the additional structure has also to be defined to fully specify a population. **Initialization** is kept simple in most EA applications: The first population is seeded by (uniformly) randomly generated individuals. However as we shall see in the next section, and succeeding chapters, there may be practical advantages to non-random initialization.

The **diversity** of a population is a measure of the number of *different* solutions present. Common diversity measures are the number of different fitness values present, the number of different phenotypes present, the number of different genotypes, and statistical measures such as entropy. Note that only one fitness value does not necessarily imply only one phenotype is present, and in turn only one phenotype does not necessarily imply only one genotype. The reverse is, however, not true: one genotype implies only one phenotype and fitness value.

As opposed to variation operators that act on the one or two parent individuals, the selection operators (parent selection and survivor selection) work at population level. In general, they take the whole current population into account. For instance, the best individual *of the given population* is chosen to seed the next generation, or the worst individual *of the given population* is chosen to be replaced by a new one. In almost all EA applications the population size is constant and does not change during the evolutionary search.

Parent Selection Mechanism

The role of **parent selection** or **mating selection** is to distinguish among individuals based on their quality to allow the better individuals to become parents of the next generation. An individual is a **parent** if it has been selected to undergo variation in order to create offspring. Together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements. In EC, parent selection is typically probabilistic. Thus, high-quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low-quality individuals are often given a small positive chance, which helps the evolutionary search avoid getting stuck in a local optimum.

Survivor Selection Mechanism

The role of **survivor selection** or **environmental selection** is to distinguish among individuals, based on their quality, to identify those that will

[4] A multiset is a set where multiple copies of an element are possible.

be used in the next generation. The survivor selection mechanism is called after the offspring of the selected parents are created. As mentioned above, in EC the population size is almost always constant, thus a choice has to be made on which individuals will be allowed in the next generation. For this reason survivor selection is also often called **replacement** or replacement strategy. This selection is usually based on their fitness values, favoring those with higher quality, although the concept of age is also frequently used. As opposed to parent selection, which is typically stochastic, survivor selection is often deterministic, for instance, ranking the unified multiset of parents and offspring and selecting the top segment (fitness biased), or selecting only from the offspring (age biased).

Variation Operators - Mutation

The role of **variation operators** is to create new individuals from old ones. In the corresponding phenotype space this amounts to generating new candidate solutions. Variation operators in EC are divided into two types based on the number of objects that they take as inputs.

Mutation, a **unary** variation operator, is applied to one genotype and delivers a (slightly) modified mutant: a **child** or **offspring** genotype. A mutation operator is always stochastic: its output – the child – depends on the outcomes of a series of random choices. It should be noted that an arbitrary unary operator is not necessarily seen as mutation. A problem-specific heuristic operator acting on one individual could be termed as mutation for being unary. However, in general mutation denotes a random, unbiased change. Thus heuristic unary operators can be distinguished from mutation in most cases.

It is important to note that variation operators are representation dependent. That is, for different representations different variation operators have to be defined. For example, if genotypes are bit-strings, then inverting a 0 to a 1 (1 to a 0) can be used as a mutation operator. However, if we represent possible solutions by tree-like structures another mutation operator is required.

Variation Operators - Recombination

Recombination (or **crossover**) is (usually) a **binary** variation operator. As the names indicate, such an operator merges information from two parent genotypes into one or two offspring genotypes. Like mutation, recombination is a stochastic operator: the choice of what parts of each parent are combined, and the way these parts are combined, depend on random events. Recombination operators with a higher arity (using more than two parents) are sometimes possible and easy to implement, but have no biological equivalent. Perhaps this is why they are not commonly used, although several studies indicate that they have positive effects on the evolution [35].

The principle behind recombination is simple – by mating two individuals with different but desirable features, it may be possible to produce an offspring that combines both of those features. This principle has a strong supporting case: it is one which has been successfully applied for millennia by breeders of plants and livestock to produce species that give higher yields or have other desirable features. Evolutionary algorithms create a number of offspring by random recombination, and accept that some will have undesirable combinations of traits, most may be no better, or even worse, than their parents, and hope that some will have improved characteristics. As with mutation, recombination operators in EAs are representation dependant. Whether to apply crossover (mutation) or not is a stochastic decision with a non-zero probability of the operator(s) not being applied.

4 A Brief Introduction to Local Search

Local search is a search method that iteratively examines the set of points in a neighborhood of the current solution and replace the current solution with a better neighbor if one exists. In this section we give a brief introduction to local search in the context of memetic algorithms. For more information there are a number of books on optimization that cover local search in more detail, such as [18]. A local search algorithm can be illustrated by the pseudocode given in Figure 2.

There are three principal components that affect the workings of this local search algorithm.

- The **pivot rule** defines the criteria for accepting an improving point. A **steepest ascent** pivot rule terminates the inner loop only after the entire neighborhood $n(i)$ has been searched, (i.e., $count = |n(i)|$). A **greedy ascent** (or *first ascent*) pivot rule terminates the inner loop as soon as an improvement is found (i.e., $((count = |n(i)|)\ or\ (best \neq i))$). In practice it is sometimes necessary to only consider a randomly drawn sample of size $N \ll |n(i)|$ if the neighborhood is too large to search.
- The **depth** of the local search defines the termination condition for the outer loop. This lies in the continuum between only one improving step being applied ($iterations = 1$) to the search continuing to local optimality where all the neighboors of a solution i have been explored but no one of them found to be better: $((count = |n(i)|)\ and\ (best = i))$. Considerable attention has been paid to studying the effect of changing this parameter within MAs [14, 25], and it can be shown to have an effect on the performance of the local search algorithm, both in terms of time taken, and in the quality of solution found. Furthermore, the impact on computational complexity of various pivot rules have been studied both in the context of local search [36, 37] and within MAs [25].

```
:
Begin
  /* given a starting solution i and a neighborhood function n */
  set best = i;
  set iterations = 0;
  Repeat Until ( depth condition is satisfied ) Do
    set count = 0;
    Repeat Until ( pivot rule is satisfied ) Do
      generate the next neighbor j ∈ n(i);
      set count = count + 1;
      If (f(j) is better than f(best)) Then
        set best = j;
      endIf
    endDo
    set i = best;
    set iterations = iterations + 1;
  endDo
End.
```

Fig. 2. Pseudocode of a local search algorithm.

- The **neighborhood generating function**, $n(i)$, defines a set of points that can be reached by the application of some move operator to the point i. The application of a neighborhood generating function can be represented as a graph $G = (v, e)$, where the set of vertices v are the points in the search space, and the edges relate to applications of the move operator; $e_{ij} \in G \iff j \in n(i)$. The provision of a scalar fitness value f defined over the search space means that we can consider the graphs defined by different move operators as fitness landscapes [14, 17, 15]. Merz and Freisleben [28] present a number of statistical measures that can be used to characterize fitness landscapes, and that have been proposed by various authors as potential measures of problem difficulty. They show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the local search, and hence of the resultant MA.

In some cases, domain-specific information may be used to guide the choice of neighborhood structure within local search algorithms. However, it has recently been shown that the optimal choice of operators can be not only instance specific within a class of problems [28, pp. 254–258], but when incorporated in an MA, it can be dependent on the state of the evolutionary search [26]. Changing the neighborhood operator during search (eg. [30]) may provide a means of progression in cases where points were locally optimal for a given neighborhood operator because a point that is locally optimal with respect to one neighborhood structure may not be with respect to another

(unless they are globally optimal). This observation has also been the guiding principle behind *variable neighborhood search* algorithm [49].

The local search method presented in Figure 2 is fairly simplistic, but local search is a central idea in most successful global search methods. The simplest of these is the so-called "multi-start local search", in which the algorithm is run repeatedly from randomly generated solutions. An elaboration on this is Iterated Local search [45], where a new search is begun from a perturbed version of the end-point of the previous one. Iterated local search attempts to traverse a succession of "nearby" local optima, which is often quite effective in practice.

Perhaps more relevant to this book are two well known heuristics based on local search, namely Tabu Search [46] and Simulated Annealing [47]. Giving a full description of these techniques is beyond the scope of this book, but in essence both modify the pivot rule. Tabu Search does so such that points in the neighborhood of the current solution which have been previously considered are not (generally) eligible to be accepted, whereas in Simulated Annealing a move to an inferior neighbor is permitted with some probability dependent on the fitness difference. Both of these have been used with noticeable success, both as heuristics in their own right, and as improvement methods within Memetic algorithms.

5 Hybridizing EAs

As suggested above, there are a number of benefits that can be achieved by combining the global search of EAs with local search or other methods for improving or refining an individual solution. In this section we give an overview of some of the theoretical and practical motivations for such hybridizations, before presenting one possible framework for Memetic Algorithms.

5.1 Motives

There are a number of factors that motivate the hybridization of evolutionary algorithms with other techniques.

- Many complex problems can be decomposed into a number of parts, for some of which exact methods (or very good heuristics) may already be available. In these cases it makes sense to use a combination of the most appropriate methods for different subproblems. In some cases this may take the form of using the EA either as a post or pre-processor for other algorithms, or incorporating instance specific knowledge into "greedy" variation operators as will be discussed in Section 7. However it is also frequently possibly to use this knowledge to define local search operators (or existing solution improvement techniques) within an evolutionary algorithm.

- Successful and efficient all-purpose "black-box" problem solvers do not exist. The rapidly growing body of empirical evidence and some theoretical results, such as the No Free Lunch (NFL) theorem [109][5] strongly support this view. From an Evolutionary Computing perspective, this implies that EAs are not the holy grail for global search. Experience suggests that in fact the competence of an EA in any given domain depends on the amount of problem-specific knowledge incorporated within it. In practice we frequently apply an evolutionary algorithm to a problem where there is a considerable amount of hard-won user experience and knowledge available. In such cases performance benefits can often arise from utilizing this information in the form of specialist operators (eg. variation and local search) and/or good solution initializations. In these cases it is commonly experienced that the combination of an evolutionary and a heuristic method – a **hybrid EA** – that somehow encapsulates domain specific information performs better than either of its "parent" algorithms alone.

- Although EAs are very good at rapidly identifying good areas of the search space (**exploration**), they are often less good at refining near-optimal solutions (**exploitation**). For example, when a GA is applied to the "One-Max" problem, near-optimal solutions are quickly found but convergence to the optimal solution is slow because the choice of which genes are mutated is random.[6] Thus EA hybrids can search more efficiently by incorporating a more systematic search in the vicinity of "good" solutions. For example, a bit-flipping hill-climber could be quickly applied within each generation for One-Max to ensure fast convergence.

- In practice, many problems have a set of constraints associated with them, and local search or other heuristics can be used as a means of "repairing" infeasible solutions generated by standard variation operators. This is often far simpler and more effective than attempting to find a specialized representation and set of variation operators which ensure the feasibility of all offspring.

- Dawkin's idea of **"memes"**[11] is often used as a motivation for hybridization. Memes can be viewed as units of "cultural transmission" in the same way that genes are the units of biological transmission. They are selected for replication according to their perceived utility or popularity, and then copied and transmitted via inter-agent communication.

[5] The NFL and its implications are still a matter of current debate, for the present we interpret it as stating that all stochastic algorithms have the same performance when averaged over all discrete problems.

[6] The One-Max problem is a binary maximization problem, where the fitness is simply the count of the number of genes set to "1".

> Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation [11, p. 192].

Since the idea of memes was first proposed by Dawkins, it has been extended by other authors (eg., [6, 13, 15, 2]). From the point of view of the study of adaptive systems as optimization techniques, memetic theory (see for example papers in [54]) provides with a rich set of tools and metaphors to work with. In the context of memetic theory an EA keeps a population of agents composed by both genotypes and memes. As in standard EA, genotypes represent solutions to a particular problem while memes represent "strategies" on how to improve those solutions. It is the memes abilities to transform a candidate solution into (hopefully) a better one that is of direct interest in the context of optimisation. The idea of memes as representing alternative improvement strategies agents can harness (implemented, for example, as distinct local searchers) is what motivated us to propose in [22] the co-evolution of memes and genes and to develop later in [25] the concept of multimeme, self-generating memetic algorithms[56],[57] and co-evolving memetic algorithms [58].

5.2 Memetic Algorithms

The most common use of hybridization within EAs, and that which fits best with Dawkin's concept of the meme, is via the application of one or more phases of improvement to individual members of the population within each generation of an EA. In the simplest design, local search is applied to individuals created by mutation or recombination. A more general form can be described by the pseudocode given in Figure 3 (see also Figure 4), although practitioners typically choose to only apply local search once to the offspring, and sometimes to avoid the use of mutation entirely when using local search.

6 Design Issues for Memetic Algorithms

So far we have discussed the rationale for the use of problem-specific knowledge or heuristics within EAs, and some possible ways in which this can be done. However, as ever we must accept the caveat that like any other technique, MAs are not some "magic solution" to optimization problems, and care must be taken in their implementation. In the sections below we briefly discuss some of the issues that have arisen from experience and theoretical reasoning.

```
:
Begin
  INITIALIZE population;
  EVALUATE each candidate;
  Repeat Until ( TERMINATION CONDITION is satisfied ) Do
    SELECT parents;
    RECOMBINE to produce offspring;
    EVALUATE offspring;
    IMPROVE offspring via Local Search;
    MUTATE offspring;
    EVALUATE offspring;
    IMPROVE offspring via Local Search;
    SELECT individuals for next generation;
  endDo
End.
```

Fig. 3. Pseudocode for a simple memetic algorithm

6.1 Lamarckianism and the Baldwin Effect

The local search methods described above assume that the current incumbent solution is always replaced by the fitter neighbor when found. Within a memetic algorithm, we can consider the local search stage to occur as an improvement, or developmental learning phase within each generation. As such, we can consider whether the changes (*acquired traits*) made to an individual should be kept, or whether the resulting improved fitness should be awarded to the original (pre-local search) member of the population.

The issue of whether acquired traits could be inherited by an individual's offspring was a major issue in nineteenth century, and Lamarck was a strong proponent of this inheritance mechanism. However, the **Baldwin effect** [59] suggests a mechanism whereby evolutionary progress can be guided towards favorable adaptation without this type of inheritance. Although modern theories of genetics strongly favor the latter viewpoint, the design of MAs can employ either Lamarckian or Baldwinian inheritance schemes. MAs are referred to as Lamarckian if the result of the local search stage replaces the individual in the population, and Baldwinian if the original member is kept, but has as its fitness the value belonging to the outcome of the local search process. In a classic early study, Hinton and Nowlan [8] showed that the Baldwin effect could be used to improve the evolution of artificial neural networks, and a number of researchers have studied the relative benefits of Baldwinian versus Lamarckian algorithms [8, 9, 10, 11, 12]. In practice, most recent work has tended to use either a pure Lamarckian approach, or a probabilistic combination of the two approaches, such that the improved fitness is always used, and the improved individual replaces the original with a given probability.

6.2 Preservation of Diversity

The problem of premature convergence, whereby the population converges around some suboptimal point, can be particularly problematic for MAs. If the local search is applied until each point has been moved to a local optimum, then this can lead to a loss of diversity within the population unless new local minima are constantly identified. Alternatively, even if local search is terminated before local optimality, an induced search space with wide basins of attractions could also result in premature convergence to the suboptimal solution at the center of a wide basin of attraction. A number of approaches have been developed to combat this problem:

- when initializing the population with known good individuals, only using a relatively small proportion of them,
- applying local search to a small fraction of the population (which helps ensure that the rest of the population is diverse),
- using recombination operators that are designed to preserve diversity,
- using multiple local searchers, where each one induces a different search space with distinct local optima (eg. [26, 12]);
- modifying the selection operator to prevent duplicates (e.g. as in CHC [31]), and
- using a fuzzy criteria, that explicitly controls diversity, as the pivot rule in the local search stage (eg. [12], 5).
- modifying the selection operator, or local search acceptance criteria, to use a Boltzmann method so as to preserve diversity (eg. III).

This last method bears natural analogies to simulated annealing [62, 47], where worsening moves can be accepted with nonzero probability to aid escape from local optima. A promising method that tackles the diversity issue explicitly is proposed in [24], where during the local search phase a less-fit neighbor may be accepted with a probability that increases exponentially as the range of fitness values in the population decreases:

$$P(accept) = \begin{cases} 1 & \text{if } \Delta E > 0, \\ e^{k* \frac{\Delta E}{F_{max} - F_{avg}}}, & \text{otherwise,} \end{cases}$$

where k is a normalization constant and we assume a maximization problem, $\Delta E = F_{\text{neighbour}} - F_{\text{original}}$.

6.3 Choice of Operators

Probably the most important factor in the design of a MA is the choice of improving heuristic or local search move operator, that is to say, the way that the set of neighboring points to be examined when looking for an improved solution is generated.

There has been a large body of theoretical and empirical analysis of the utility of various statistical measures of landscapes for predicting problem difficulty. The interested reader can find a good summary in [64]. Merz and Freisleben [28] consider a number of these measures in the context of memetic algorithms, and show that the choice of move operator can have a dramatic effect on the efficiency and effectiveness of the local search, and hence of the resultant MA.

One recent result of particular interest to the practitioner is Krasnogor's formal proof that, in order to reduce the worst-case run times, it is necessary to choose a local search method whose move operator is not the same as those of the recombination and mutation operators [25]. This formalizes the intuitive point that within a MA recombination, and particularly mutation, have valuable roles in generating points that lie in different basins of attraction with respect to the local search operator. This diversification is best done either by an aggressive mutation rate, or preferably by the use of a variation operators that have different neighborhood structures.

In general then, it is worth giving careful consideration to the choice of move operators used when designing a MA: for example, using 2-opt for a TSP problem might yield better improvement if not used in conjunction with the "inversion" mutation operator which picks a subtour at random and reverses it. The reason for that is that a genotypic inversion induces (a subspace of) the phenotypic effect of the 2-exchange move operator which is at the heart of 2-opt local searcher.

In some cases, domain-specific information may be used to guide the choice of neighborhood structure within the local search algorithms. However, as we noted earlier, the optimal choice of operators can be not only instance specific within a class of problems but also dependant on the state of the evolutionary search.

One simple way to surmount these problems is the use of *multiple* local search operators in tandem. An example of this can be seen in [30], where a range of problem specific move operators, such as local stretches, rotations and reflections, each tailored to different stages of the protein folding process, are used for a protein structure prediction problem within the context of what is called a *multimemetic algorithm* [26].

The use of a set of possible local search strategies is analogous to Dawkin's memes. The extension of this approach to allow the adaptation of the local search "memes" in the form of a coevolving population, and the implications for search is currently under way in different research groups [22, 65, 22, 37, 68, 58, 69, 56, 57].

6.4 Use of Knowledge

A final point that might be taken into consideration when designing a MA concerns the use and reuse of knowledge gained during the optimization process. One possible hybridization that explicitly uses knowledge about points

already searched to guide optimization is with **tabu search** [46]. In this algorithm a "tabu" list of visited points is maintained, which the algorithm is forbidden to return to. Such methods appear to offer promise for maintaining diversity. Similarly, it is easy to imagine extensions to the Boltzmann acceptance/selection schemes that utilize information about the spread of genotypes in the current population, or even past populations, when deciding whether to accept new solutions.

6.5 Specific Considerations for Continuous Domains

The design of MAs for continuous domains is complicated by several factors. Effective search requires the use of different *search scales* for global and local search. It is not always possible to determine whether a solution is locally optimal. Relatively long local searches may be needed to ensure convergence to local optima (especially if gradient information is unavailable). Although many different local search methods have been developed, they are general methods and thus it is not clear whether any given local search method is effective for a particular application.

Because of these considerations, the design of effective MAs for continuous domains can be quite different than for combinatorial problems. For example, in combinatorial domains it is not unusual to apply local search until a locally optimal solution is found. However, it is often unrealistic to assume that local search methods can quickly identify local minima within a continuous domain. This is often the case when applying derivative-free methods (e.g. the Nelder-Mead simplex method), but it may also be true when derivative information is available. Thus it is generally the case that local search is truncated based on a target balance between global and local search. Specifically, two main strategies have been used to achieve such a balance: (1) truncate local searches after a given number of iterations (or fitness evaluations) and (2) apply local search infrequently (e.g. with a fixed probability).

Although these hybridization strategies are quite effective in practice, they can make it difficult to ensure convergence for these MAs. Although general conditions on the mutation and recombination operators can be enforced to ensure global convergence [70], these convergence results provide little insight into the efficacy of local search. Gradient-based methods can be applied to generate stationary-points (using first-order information) or locally-optimal points (using second-order information), assuming that local search is not truncated after a given number of iterations. However, in many applications derivative-free methods are applied for which the search is truncated. To our knowledge, MAs based on evolutionary pattern search is the only class of MAs for which the convergence of tandem derivative-free local searches within the MA can be ensured [71].

7 Other Hybridization Possibilities

Although our working definition of MAs has been restricted to those methods that incorporate some form of improvement mechanism acting on the output of the evolutionary variation operators, there are a number of other ways in which an EA or MA can be used in conjunction with other operators and/or domain-specific knowledge. This is illustrated in Figure 4.

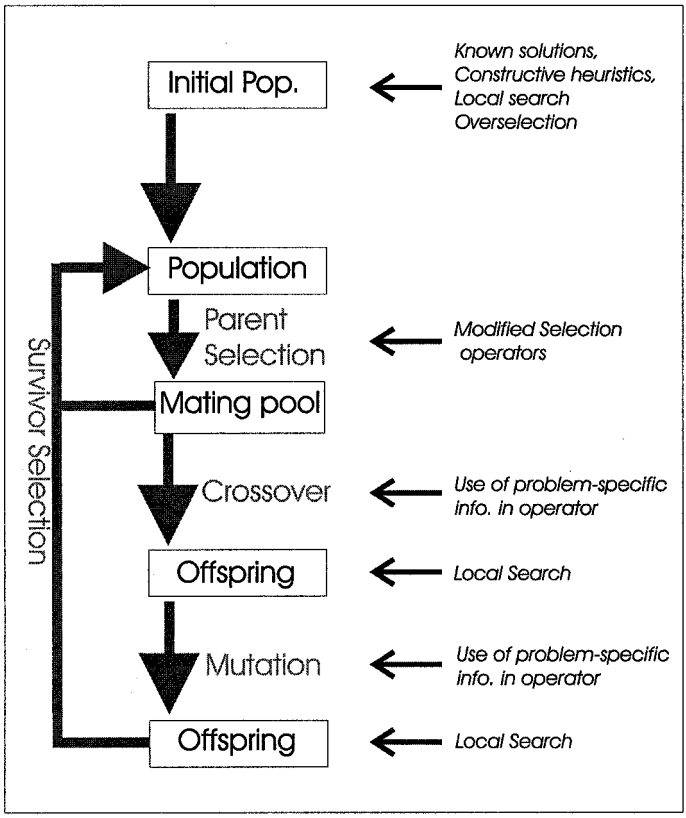

Fig. 4. Possible places to incorporate knowledge or other operators within a single generation.

7.1 Intelligent Initialization

The most obvious way in which existing knowledge about the structure of a problem or potential solutions can be incorporated into an EA is in the initialization phase. In many cases the EA will make rapid initial progress,

which raises questions about the value of expending effort creating a good initial population, however starting the EA by using existing solutions can offer interesting benefits:

1. It is possible to avoid "reinventing the wheel" by using existing solutions. Preventing waste of computational efforts can yield increased efficiency (speed).
2. A nonrandom initial population can direct the search into particular regions of the search space that contain good solutions. Biasing the search can result in increased effectiveness (quality of end solution).
3. All in all, a given total amount of computational effort divided over heuristic initialization and evolutionary search might deliver better results than spending it all on "pure" evolutionary search, or an equivalent multistart heuristic.

There are a number of possible ways in which the initialization function can be changed from simple random creation, such as:

- **Seeding** the population with one or more previously known good solutions arising from other techniques.
- In **selective initialization** a large number of random solutions are created and then the initial population is selected from these. Bramlette [72] suggests that this should be done as a series of N k-way tournaments rather than by selecting the best N from $k \cdot N$ solutions. Other alternatives include selecting a set based not only on fitness but also on diversity so as to maximize the coverage of the search space.
- Performing a local search starting from each member of initial population, so that the initial population consists of a set of points that are locally optimal with respect to some move operator.
- Using one or more of the above methods to identify one (or possibly more) good solutions, and then cloning them and applying mutation at a high rate (**mass mutation**) to produce a number of individuals in the vicinity of the start point.

These methods have been tried and have exhibited performance gains for certain problems. However, the important issue of providing the EA with sufficient diversity for evolution to occur must also be considered. In [73] Surry and Radcliffe examined the effect of varying the proportion of the initial population of a GA that was derived from known good solutions. Their conclusions were:

- The use of a small proportion of derived solutions in the initial population aided genetic search.
- As the proportion was increased, the *average* performance improved.
- The *best* performance came about from a more random initial population.

In other words, as the proportion of solutions derived from heuristics used increased, so did the mean performance, but the variance in performance

decreased. This meant that there were not the occasional really good runs resulting from the EA searching completely new regions of space and coming up with novel solutions. For a certain type of problems, such as design problems, this is an undesirable property.

7.2 Hybridization During Genotype to Phenotype Mapping

A widely used hybridization of memetic algorithms with other heuristics is during the genotype–phenotype mapping $\mathcal{M}$ prior to evaluation. This approach, where the EA is used to provide the inputs controlling the application of another heuristic, is frequently used and similar approaches have been used to great effect for timetabling and scheduling problems [74], and in the "sector first–order second" approach to the vehicle routing problem [75].

There is a common thread to all of these approaches, which is to make use of existing heuristics and domain information wherever possible. The role of the EA is often that of enabling a less biased application of the heuristics, or of problem decomposition, so as to permit the use of sophisticated, but badly scaling heuristics when the overall problem size would preclude their use.

7.3 Hybridization Within Variation Operators: Intelligent Crossover and Mutation

A number of authors have proposed so-called "intelligent" variation operators, which incorporate problem- or instance-specific knowledge. To give a simple example, if a binary-coded GA is used to select features for use in another classification algorithm, one might attempt to bias the search towards more compact features sets via the use of a greater probability for mutating from the allele value "use" to "don't use" rather than vice versa. A related approach can be seen in [76], where genes encode for microprocessor instructions, which group naturally into sets with similar effects. The mutation operator was then biased to incorporate this expert knowledge, so that mutations were more likely to occur between instructions in the same set than between sets.

A slightly different example of the use of problem-specific (rather than instance-specific) knowledge can be seen in the modified one-point crossover operator used for protein structure prediction in [77]. Here the authors realized that the heritable features being combined by recombination were folds, or fragments of three-dimensional structure. A property of the problem is that during folding protein structures can be free to rotate about peptide bonds. The modified operator made good use of this knowledge by explicitly testing all the possible different orientations of the two fragments, (accomplished by trying all the possible allele values in the gene at the crossover point) in order to find the most energetically favorable. If no feasible conformation was found, then a different crossover point was selected and the process repeated. This could be seen as a simple example of the incorporation of a local search phase into the recombination operator, but in practice the nature of the models

used is such that generally these approaches only need to consider partial solutions when deciding whether an offspring is feasible. Note that this should be distinguished from the simpler "crossover hill-climber" proposed in [15], in which all of the l-1 possible offspring arising from one-point crossover are constructed and the best chosen.

Operators can be modified in a complex manner to incorporate highly specific heuristics, which makes use of instance-specific knowledge. A good example of this is Merz and Friesleben's distance-preserving crossover (DPX) operator for the TSP [78]. This operator has two motivating principles: making use of instance specific knowledge, whilst at the same time preserving diversity within the population to prevent premature convergence. Diversity is maintained by ensuring that the offspring inherits all of the edges common to both parents, but none of the edges that are present in only one parent. The "intelligent" part of the operator comes from the use of a nearest-neighbor heuristic to join together the subtours inherited from the parents, thus explicitly exploiting instance-specific edge length information. It is easy to see how this type of scheme could be adapted to other problems, via the use of suitable heuristics for completing the partial solutions after inheritance of the common factors from both parents.

It should be noted that under our working definition of MAs, the use of such "intelligent" operator within an EA does not generally on its own constitute a MA, since they use instance-specific knowledge to guide the construction of *partial* solutions. This can be contrasted with the use of local search acting on offspring, where a neighborhood of *complete* solutions is examined and an improved solution accepted.

8 Conclusions

In this chapter we gave a gentle introduction to Memetic Evolutionary Algorithms and role they play as complements to pure Evolutionary Algorithms and pure Local Search. We briefly discussed the historical context of MAs, and we gave the motivation behind the use and research on this important brand of global-local search hybrids. We also mentioned some of the design principles a practitioner needs to take into consideration when designing Memetic Algorithms for new domains.

References

1. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Efective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany (2000)

2. Vavak, F., Fogarty, T., Jukes, K.: A genetic algorithm with variable range of local search for tracking changing environments. In Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P., eds.: Proceedings of the 4th Conference on Parallel Problem Solving from Nature. Number 1141 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (1996) 376–385
3. Knowles, J., Corne, D.: A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective d-msat problem. In: Gecco-2001 Workshop Program. (2001) 162–167
4. Moscato, P.: Memetic algorithms' home page, visited july 2003: http://www.densis.fee.unicamp.br/~moscato/memetic_home.html (2003)
5. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program Report 826, Caltech, Caltech, Pasadena, California (1989)
6. Hinton, G., Nowlan, S.: How learning can guide evolution. Complex Systems **1** (1987) 495–502
7. Bull, L., Fogarty, T.: An evolutionary strategy and genetic algorithm hybrid: An initial implementation and first results. In Fogarty, T., ed.: Evolutionary Computation: Proceedings of the 1994 AISB Workshop on Evolutionary Computing, Springer, Berlin, Heidelberg, New York (1994) 95–102
8. Houck, C., Joines, J., Kay, M., Wilson, J.: Empirical investigation of the benefits of partial Lamarckianism. Evolutionary Computation **5** (1997) 31–60
9. Mayley, G.: Landscapes, learning costs and genetic assimilation. Evolutionary Computation **4** (1996) 213–234
10. Turney, P.: How to shift bias: lessons from the Baldwin effect. Evolutionary Computation **4** (1996) 271–295
11. Whitley, L., Gordon, S., Mathias, K.: Lamarkian evolution, the Baldwin effect, and function optimisation. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Proceedings of the 3rd Conference on Parallel Problem Solving from Nature. Number 866 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (1994) 6–15
12. Whitley, L., Gruau, F.: Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evolutionary Computation **1** (1993) 213–233
13. Hart, W.: Adaptive Global Optimization with Local Search. PhD thesis, University of California, San Diego (1994)
14. Krasnogor, N.: Studies in the Theory and Design Space of Memetic Algorithms. PhD thesis, University of the West of England (2002)
15. Land, M.: Evolutionary Algorithms with Local Search for Combinatorial Optimization. PhD thesis, University of California, San Diego (1998)
16. Moscato, P.: Problemas de Otimizacão NP, Aproximabilidade e Computacão Evolutiva:Da Prática à Teoria. PhD thesis, Universidade Estadual de Campinas,Brasil (2001)
17. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Springer, Berlin, Heidelberg, New York (2003)
18. Aarts, E., Lenstra, J., eds.: Local Search in Combinatorial Optimization. Discrete Mathematics and Optimization. Wiley, Chichester, UK (1997)
19. Fogel, D., ed.: Evolutionary Computation: the Fossil Record. IEEE Press, Piscataway, NJ (1998)

20. Fogel, L., Owens, A., Walsh, M.: Artificial intelligence through a simulation of evolution. In Callahan, A., Maxfield, M., Fogel, L., eds.: Biophysics and Cybernetic Systems. Spartan, Washington DC (1965) 131–156
21. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence through Simulated Evolution. Wiley, Chichester, UK (1966)
22. De Jong, K.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan (1975)
23. Holland, J.: Genetic algorithms and the optimal allocation of trials. SIAM J. of Computing **2** (1973) 88–105
24. Holland, J.: Adaption in Natural and Artificial Systems. MIT Press, Cambridge, MA (1992) 1st edition: 1975, The University of Michigan Press, Ann Arbor.
25. Rechenberg, I.: Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart (1973)
26. Schwefel, H.P.: Evolution and Optimum Seeking. Wiley, New York (1995)
27. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford, UK (1996)
28. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol (2000)
29. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Evolutionary Computation 2: Advanced Algorithms and Operators. Institute of Physics Publishing, Bristol (2000)
30. Eiben, A., Michalewicz, Z., eds.: Evolutionary Computation. IOS Press (1998)
31. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer, Berlin, Heidelberg, New York (1996)
32. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic Programming: An Introduction. Morgan Kaufmann, San Francisco (1998)
33. Koza, J.: Genetic Programming. MIT Press, Cambridge, MA (1992)
34. Koza, J.: Genetic Programming II. MIT Press, Cambridge, MA (1994)
35. Eiben, A.: Multiparent recombination. [28] chapter 33.7 289–307
36. Johnson, D., Papadimitriou, C., Yannakakis, M.: How easy is local search. Journal of Computer And System Sciences **37** (1988) 79–100
37. Yannakakis, M.: Computational complexity. In Aarts, E., Lenstra, J., eds.: Local Search in Combinatorial Optimization, John Wiley & Sons Ltd. (1997) 19–55
38. Weinberger, E.D.: Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. Biological Cybernetics **63** (1990) 325–336
39. Stadler, P.F.: Towards a Theory of Landscapes. In Lopéz-Peña, R., Capovilla, R., García-Pelayo, R., Waelbroeck, H., Zertuche, F., eds.: Complex Systems and Binary Networks. Volume 461 of Lecture Notes in Physics., Berlin, New York, Springer Verlag (1995) 77–163 SFI preprint 95-03-030.
40. Jones, T.: Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, The University of New Mexico, Albuquerque, NM (1995)
41. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw Hill, London (1999) 245–260
42. Krasnogor, N., Smith, J.: Emergence of profitable search strategies based on a simple inheritance mechanism. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., Burke,

E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, San Francisco (2001) 432–439
43. Krasnogor, N., Blackburne, B., Burke, E., Hirst, J.: Multimeme algorithms for protein structure prediction. [12] 769–778
44. Hansen, P., Mladenovič, N.: An introduction to variable neighborhood search. In Voß, S., Martello, S., Osman, I., Roucairol, C., eds.: Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Proceedings of MIC 97 Conference. Kluwer Academic Publishers, Dordrecht, The Netherlands (1998)
45. Lourenco, H.R., Martin, O., Stutzle, T.: Iterated local search. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, MA (2002) 321–353
46. Glover, F.: Tabu search: 1. ORSA Journal on Computing **1** (1989) 190–206
47. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated anealing. Science **220** (1983) 671–680
48. Wolpert, D., Macready, W.: No Free Lunch theorems for optimisation. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
49. Dawkins, R.: The Selfish Gene. Oxford University Press, Oxford, UK (1976)
50. Cavalli-Sforza, L., Feldman, M.: Cultural Transmission and Evolution: A Quatitative Approach. Princeton University Press, Princeton, NJ. (1981)
51. Durham, W.: Coevolution: Genes, Culture and Human Diversity. Stanford University Press (1991)
52. Gabora, L.: Meme and variations: A computational model of cultural evolution. In L.Nadel, Stein, D., eds.: 1993 Lectures in Complex Systems. Addison Wesley (1993) 471–494
53. Blackmore, S.: The Meme Machine. Oxford University Press, Oxford, UK (1999)
54. of Memetics. Advisory Board:S.Blackmore, J., G.Cziko, R.Dawkins, D.Dennett, L.Gabora, D.Hull., eds.: Journal of Memetics: Evolutionary Models of Information Transmission. (http://jom-emit.cfpm.org/)
55. Krasnogor, N.: Co-evolution of genes and memes in memetic algorithms. In Wu, A., ed.: Proceedings of the 1999 Genetic And Evolutionary Computation Conference Workshop Program. (1999)
56. Krasnogor, N.: Self-generating metaheuristics in bioinformatics: The protein structure comparison case. Genetic Programming and Evolvable Machines. Kluwer academic Publishers **5** (2004) 181–201
57. Krasnogor, N., Gustafson, S.: A study on the use of "self-generation" in memetic algorithms. Natural Computing **3** (2004) 53–76
58. Smith, J.: Co-evolving memetic algorithms: A learning approach to robust scalable optimisation. [1] 498–505
59. Baldwin, J.: A new factor in evolution. American Naturalist **30** (1896)
60. Krasnogor, N., Pelta, D.: Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid. In Verdegay, J., ed.: Fuzzy Sets based Heuristics for Optimization, Springer (2002)
61. Eshelman, L.: The CHC adaptive search algorithm: how to have safe search when engaging in non-traditional genetic recombination. In Rawlins, G., ed.: Foundations of Genetic Algorithms, Morgan Kaufmann, San Francisco (1990) 263–283
62. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines. Wiley, Chichester, UK (1989)

63. Krasnogor, N., Smith, J.: A memetic algorithm with self-adaptive local search: TSP as a case study. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.G., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, San Francisco (2000) 987–994
64. Kallel, L., Naudts, B., Reeves, C.: Properties of fitness functions and search landscapes. In Kallel, L., Naudts, B., Rogers, A., eds.: Theoretical Aspects of Evolutionary Computing. Springer, Berlin, Heidelberg, New York (2001) 175–206
65. Bull, L., Holland, O., Blackmore, S.: On meme–gene coevolution. Artificial Life **6** (2000) 227–235
66. Krasnogor, N., Gustafson, S.: Toward truly "memetic" memetic algorithms: discussion and proofs of concept. In Corne, D., Fogel, G., Hart, W., Knowles, J., Krasnogor, N., Roy, R., Smith, J., Tiwari, A., eds.: Advances in Nature-Inspired Computation: The PPSN VII Workshops, Reading, UK, PEDAL (Parallel, Emergent & Distributed Architectures Lab), University of Reading (2002) 9–10
67. Smith, J.: Co-evolution of memetic algorithms: Initial investigations. [12] 537–548
68. Smith, J.: The co-evolution of memetic algorithms for protein structure prediction. In Corne, D., Fogel, G., Hart, W., Knowles, J., Krasnogor, N., Roy, ., Smith, J., Tiwari, A., eds.: Advances in Nature-Inspired Computation: The PPSN VII Workshops, Reading, UK, PEDAL (Parallel, Emergent & Distributed Architectures Lab), University of Reading (2002) 14–15
69. Smith, J.: Protein structure prediction with co-evolving memetic algorithms. [1] 2346–2353
70. Rudolph, G.: Convergence of evolutionary algorithms in general search spaces. [82] 50–54
71. Hart, W., DeLaurentis, J., Ferguson, L.: On the convergence of an implicitly self-adaptive evolutionary algorithm on one-dimensional unimodal problems. IEEE Trans Evolutionary Computation (to appear) (2003)
72. Bramlette, M.: Initialization, mutation and selection methods in genetic algorithms for function optimization. In Belew, R., Booker, L., eds.: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco (1991) 100–107
73. Surry, P., Radcliffe, N.: Innoculation to initialise evolutionary search. In T.C.Fogarty, ed.: Evolutionary Computing: Proceedings of the 1996 AISB Workshop, Springer, Berlin, Heidelberg, New York (1996) 269–285
74. Hart, E., Ross, P., Nelson, J.: Solving a real-world problem using an evolving heuristically driven schedule builder. Evolutionary Computation **6** (1998) 61–81
75. Thangiah, S., Vinayagamoorty, R., Gubbi, A.: Vehicle routing and time deadlines using genetic and local algorithms. [81] 506–515
76. Smith, J., Bartley, M., Fogarty, T.: Microprocessor design verification by two-phase evolution of variable length tests. In: Proceedings of the 1997 IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ (1997) 453–458
77. Unger, R., Moult, J.: A genetic algorithm for 3D protein folding simulations. [81] 581–588
78. Friesleben, B., Merz, P.: A genetic local search algorithm for solving the symmetric and assymetric travelling salesman problem. [82] 616–621

79. Guervos, J.M., Adamidis, P., Beyer, H.G., Fernandez-Villacanas, J.L., Schwefel, H.P., eds.: Proceedings of the 7th Conference on Parallel Problem Solving from Nature. In Guervos, J.M., Adamidis, P., Beyer, H.G., Fernandez-Villacanas, J.L., Schwefel, H.P., eds.: Proceedings of the 7th Conference on Parallel Problem Solving from Nature. Number 2439 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, New York (2002)
80. 2003 Congress on Evolutionary Computation (CEC 2003). In: 2003 Congress on Evolutionary Computation (CEC 2003), IEEE Press, Piscataway, NJ (2003)
81. Forrest, S., ed.: Proceedings of the 5th International Conference on Genetic Algorithms. In Forrest, S., ed.: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco (1993)
82. Proceedings of the 1996 IEEE Conference on Evolutionary Computation. In: Proceedings of the 1996 IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ (1996)