Memetic Algorithms

Natalio Krasnogor

Abstract Memetic Algorithms have become one of the key methodologies behind solvers that are capable of tackling very large, real-world, optimisation problems. They are being actively investigated in research institutions as well as broadly applied in industry. In this chapter we provide a pragmatic guide on the key design issues underpinning Memetic Algorithms (MA) engineering. We begin with a brief contextual introduction to Memetic Algorithms and then move on to define a Pattern Language for MAs. For each pattern, an associated design issue is tackled and illustrated with examples from the literature. In the last section of this chapter we "fast forward" to the future and mention what, in our mind, are the key challenges that scientistis and practitioner will need to face if Memetic Algorithms are to remain a relevant technology in the next 20 years.

1 Introduction

Memetic Algorithms (MAs) was the name given by P.A. Moscato[102] to a class of stochastic global search techniques that, broadly speaking, combine within the framework of Evolutionary Algorithms (EAs) the benefits of problem-specific local search heuristics and multi-agent systems. MAs have been successfully applied to a wide range of domains that cover problems in combinatorial optimisation, e.g. [120, 23, 39, 54, 28, 26, 134, 46], continous optimisation, e.g. [53, 108, 101], dynamic optimisation [137, 25, 138], multi-objective optimization [91, 59, 61], etc. It could be argued that, unlike other nature-inspired algorithms such as Ant Colony Optimisation [35], Simulated Annealing[68], Neural Networks[95], Evolutionary Algorithms[56], etc, Memetic Algorithms lack, at their core, a clear natural

Natalio Krasnogor
Interdisciplinary Optimisation Laboratory, The Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science, University of Nottingham, United Kingdom, e-mail: Natalio.Krasnogor@Nottingham.ac.uk

metaphor. Whether this is a strength or a weakness of the paradigm is a discussion for another paper and in this chapter we opted instead to focus first on a pragmatic software engineering presentation of this remarkably malleable search technology and then, near the end of the chapter, to argue that there is indeed a potentially powerful nature-inspired metaphor that could lead to important new breakthroughs in the field.

Early in the history of EAs' application to real-world problems, it became aparent that a *canonical GA*, namely one using a simple binary representation, n-point crossover, bitwise mutation and fitness proportionate selection, could not possibly compete with tailor made algorithms. This empirical observation resonated well with theoretical (and experimental) studies on the so called "Baldwin effect" and on "Lamarckian evolution" [55, 14, 58, 94, 136, 139, 140] that focused on how learning could affect the process of evolution. Thus, it became apparent that EA's global search dynamics ought to be complemented with local search refinement provided by a suitable hybridisation using problem specific solvers including heuristics, approximate and exact algorithms. Moreover, further theoretical results such as [141] (and similar subsequent work) debunked the idea that effective and efficient "black box" general problem solvers were attainable, and hence gave further impetus to the school of thought that supported, as an essential methodological component, the incorporation of problem (or domain) specific information in EAs[1] . Domain-specific knowledge was thus added to the EA framework by means of specialised crossover and mutation operators, sophisticated problem specific representations, smart population initialisation, complex fitness functions, (closer to the spirit of MAs) local search heuristics and, when available, approximate and exact methods. More recently, R. Dawkin's concept of "memes" [33] has been gathering pace within the Memetic Algorithms literature as they can be thought of as representing "evolvable" strategies for problem solving thus breaking the mould of a fixed and static domain knowledge captured once during the design of the MAs and left untouched afterwards. Thus Dawkin's memes, and their extensions ([27, 38, 41, 9]), as evolvable search strategies [70, 71, 73, 77, 128, 19] provide a critical link to the possibility of open-ended combinatorial and/or continuous problem solving.

Software development is a process of knowledge acquisition[3] and the development of successful Memetic Algorithms is no different. The popularity behind MAs is more closely related to the relative ease by which a reasonably good solver can be implemented than to any fundamental advantage over other optimisation techniques such as Tabu Search or Simulated Annealing (to name but two). Indeed, any successful nature-inspired search method owes it's popularity not to an intrinsic problem solving feature, which might be absent from a competing method, but rather to the fact that, in spite of obvious design flaws (e.g. large number of parameters, lack of operational theory for their use, etc), they help to structure around them a healthy research and practice milieu. That is, nature-inspired search methods are computational "research programmes", or "research paradigms", in their own right

[1] Please note that everything we have said so far for Evolutionary Algorithms would also be applicable to other search frameworks such as Tabu Search, Simmulated Annealing or Ant Colony Optimisation, etc.

[85]. Thus the question of what are the key components of the Memetic Algorithms research paradigm takes center stage. The literature has a large number of papers in which a variety of methods are classified as Memetic Algorithms. Thus, although the large majority of Memetic Algorithms are instances of Evolutionary Algorithms-Local Search hybrids, numerous MAs are derived from other metaheuristics, e.g., Ant Colony Optimisation (ACO) [88], Particle Swarm [90], Artificial Immune Systems (AIS)[142], etc. What all of these implementations have in common is a *carefully choreographed interplay between (stochastic) global search and local search strategies*. In the remaining parts of this chapter, we will overview some of the key implementation strategies that, over the course of the years, have (re)appeared in the form of tried and tested algorithmic design solutions to the ubiquitous problem of how to successfully orchestrate global and local search methods in complex search spaces.

2 A Pattern Language for Memetic Algorithms

In [2] C. Alexander and colleagues introduced, within the context of architecture and urban planning, the concept of “Patterns” and “Pattern Languages”:

> In this book, we present one possible pattern language,... The elements of this language are entities called patterns. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

A pattern language is then defined as a collection of interrelated patterns, with each and every one of them expressed in a concise, clear and uniform format. The content of a pattern includes at least the following elements [43]:

- *The pattern name*, which is a concise handler to refer to both a specific problem and a tried and tested solution. By having a carefully selected set of pattern names (e.g. Selection Mechanism, Crossover Strategy, Exploration Strategy, Diversification Plan, etc) the pattern language, i.e. the vocabulary, of – in our case nature-inspired paradigms, e.g. Memetic Algorithms – is enriched. Critically, a small increase in the size of a vocabulary creates a rich and expressive combinatorial explosion of patterns’ compositions thus opening the road for substantial research and practical experimentation. The importance of this observation will become apparent once we describe Self-Generating Memetic Algorithms.
- *The problem statement* depicting the situation in which the pattern is best applied, that is, the problem that the pattern attempts to provide a solution to, e.g., maintaining pareto front diversity, etc. The problem statement might also contain a set of constraints describing situations where the pattern should not be applied or, symmetrically, conditions that must be fulfilled before the pattern can be used.
- *The solution*, in turn, provides a template on how to approach the solution of the problem to which the pattern is applied. The description in this section is not

prescriptive but rather qualitative. As emphasized by Alexander et al. [2], one might reuse a solution under myriads of different shapes, yet the essential core of all those implementations should be easily distinguishable and invariant.

- *The consequences* of applying the pattern. There are no free lunches, hence even when a pattern might be the best (or perhaps only) solution to a given problem, its application might lead to a series of trade-offs. The more explicit and clearly stated these are, the more clear and precise the pattern language as a whole will be. For example, a pattern that calls for the reinitialisation of a population due to a diversity crisis might carry with it, as obvious collateral damage, certain loss of information. Thus, by employing a Reinitialisation pattern one might be forced to utilise a pattern that safeguards partial solutions.
- *Representative Examples* briefly mentioning cases where the pattern has been used.

Thus a collection of well defined patterns, i.e. a rich pattern language, substantially enhances our ability to communicate solutions to recurring problems without the need to discuss specific implementation details. The pattern language thus serves the dual purpose of being both a taxonomy of problems and a catalogue of solutions. In this chapter we will provide a series of patterns that will be defined as per the tableaux that appears above. A reader interested in, for example, finding out about diversity handling strategies for MAs, irrespective of which underlying framework (e.g. Evolutionary, Ant Colony, Artificial Immune System, etc) the MA is being implemented in, can quickly scan the various patterns in the pattern language catalogue, identify the one related to diversity strategies and rapidly gain an idea of the tried and tested approaches, the pattern's motivation and consequences. Furthermore, the reader could then refer to the mentioned literature for concrete, detailed codes and methods.

It has been argued [30] that design patterns are seldom created but rather they are discovered through a process of datamining the source code base and literature base for reusable solutions to recurring problems. In this spirit, and before describing a specific pattern language for Memetic Algorithms, we overview the pseudocodes and flowcharts of some representative MA instances. We resort to reporting these algorithms in exactly the same form found in the originating publication as to emphasise both the *invariants* in their architecture as well as the variety of "decorations" found in the many implementations of Memetic Algorithms. Other examples of "in the wild" MAs can be found in [82].

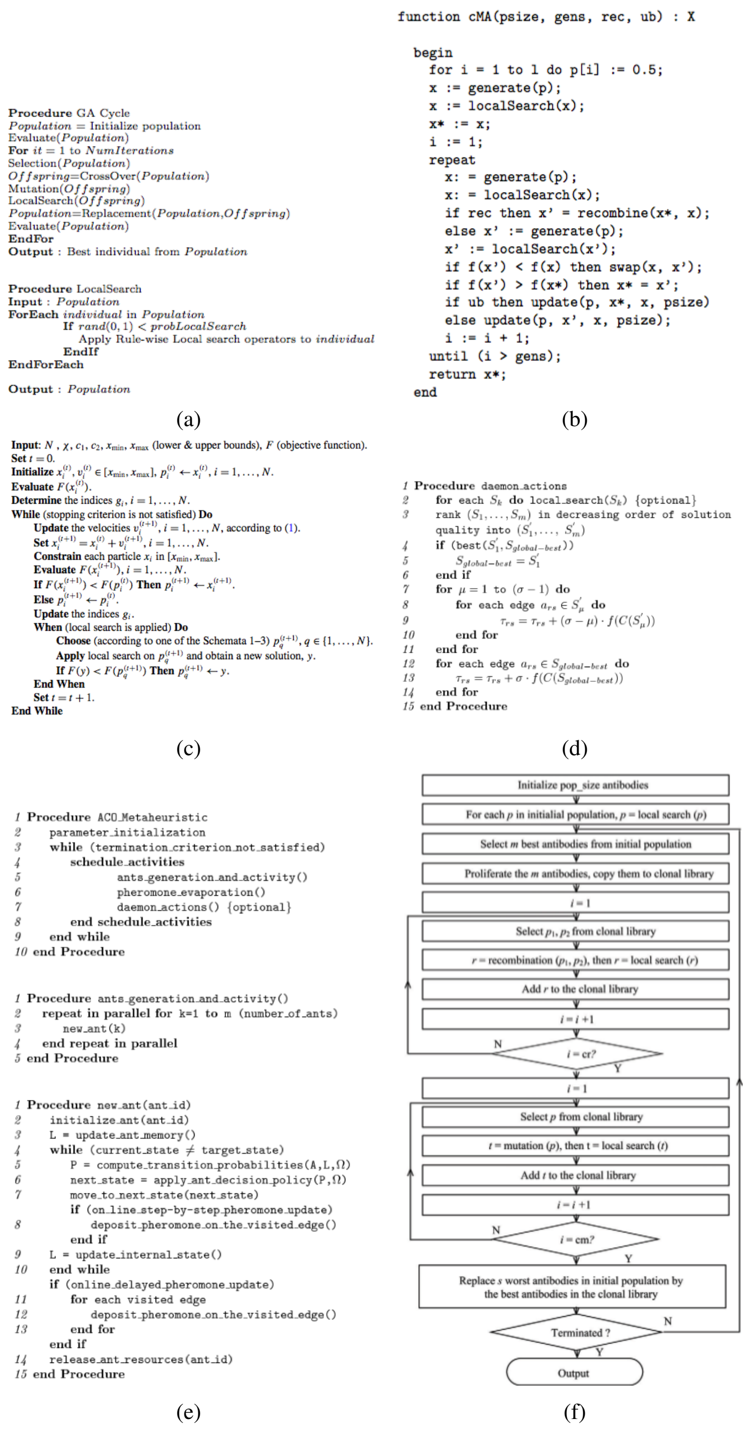
Memetic improvements have been used in, e.g., Learning Classifier Systems [4] with the top level pseudocode shown in Fig. 1(a). An Estimation of Distribution-like MA, a Compact Memetic Algorithm[96] is shown in 1(b) (for a more recent EDA-MA see [37]) while a Memetic Particle Swarm Optimisation [115] pseudocode is depicted in 1(c). Figures 1(d) and 1(e) show a generic pseudocode of an Ant Colony Optimisation based MA[31] and its explicit use of solution refining strategies (in the form of local search methods) respectively. An example of an Immune System inspired Memetic Algorithm's [142] flowchart is shown in figure 1(f). The key invariant property that is present in the architecture of all these Memetic Algorithms is the combination of a search mechanism operating over (in principle) the entire search space with other search operators focusing on local regions of these search spaces. This key invariant holds true regardless of the nature-inspire paradigm the Memetic Algorithm is derived from or whether it is meant to solve an NP-hard combinatorial problem or a highly complex (e.g. multimodal, nonlinear, multi-dimensional) continuous one.

We argue that the key problem that is addressed by Memetic Algorithms is the balance between global and local search, in other words, the strategy that different nature-inspired paradigms (e.g. ACO, AIS, etc) might need to implement as to benefit from a successful tradeoff between exploration and exploitation. Thus we can define the first top-level pattern as:

NAME: Memetic Algorithm Pattern (MAP)[1]

- *Problem statement:* A Memetic Algorithm provides solution patterns for the ubiquitous problem of how to successfully orchestrate a balanced tradeoff between exploring a search space and exploiting available (partial) solutions. It is suitable for the solution of complex problems where standard and efficient (i.e. approximation algorithms, exact algorithms, etc) methods do not exist. The Memetic Algorithm is said to explore the search space through a "global" search technique while exploitation is achieved through "local" search.
- *The solution:* This pattern relies on finding, for a given problem domain, an adequate instantiation of exploration and exploitation. Exploration is performed by "global search" methods usually implemented by means of a population-based nature-inspired method such as Evolutionary Algorithms, Ant Colony Optimisation, Artificial Immune Systems, etc. Exploitation is commonly done through the use of local search methods and domain specific heuristics. The global scale exploration is achieved by, e.g., keeping track of multiple solutions or by virtue of specific "jump" operators that are able to connect distant regions of the search space. The local scale exploitation focuses the search on the vicinity of a given candidate solution.
- *The consequences:* Hybridising a global search method of any kind with local search and/or domain specific heuristics usually results in better end-results but this comes at the expense of increased computational time. The correct tradeoff between exploration and exploitation must be such that were the global searcher given the same total CPU "budget" as the Memetic Algorithm then its solutions would still be worse than those derived from the MA. Needless to say, if the local searcher by itself, or through a naive multi-start shell could achieve the same quality of results as the Memetic Algorithm then the global searcher becomes irrelevant. Another likely consequence is that local search and domain specific heuristics usually result in premature convergence (also called diversity crisis).
- *Examples:* Papers [31, 142, 115, 37, 4] report on the use of Memetic Algorithms under a variety of nature-inspired incarnations.

The MAP can be refined through a variety of Template Patterns (TP)[43], which allow for the definition of the *skeleton* of an algorithm, method or protocol, through deferring problem specific details to subclasses. In this way, through a judicious



```
Procedure GA Cycle
Population = Initialize population
Evaluate(Population)
For $it = 1$ to NumIterations
Selection(Population)
Offspring=CrossOver(Population)
Mutation(Offspring)
LocalSearch(Offspring)
Population=Replacement(Population,Offspring)
Evaluate(Population)
EndFor
Output : Best individual from Population


Procedure LocalSearch
Input : Population
ForEach individual in Population
        If $rand(0,1) < probLocalSearch$
            Apply Rule-wise Local search operators to individual
        EndIf
EndForEach

Output : Population
```

(a)

```
function cMA(psize, gens, rec, ub) : X

  begin
    for i = 1 to l do p[i] := 0.5;
    x := generate(p);
    x := localSearch(x);
    x* := x;
    i := 1;
    repeat
      x: = generate(p);
      x: = localSearch(x);
      if rec then x' = recombine(x*, x);
      else x' := generate(p);
      x' := localSearch(x');
      if f(x') < f(x) then swap(x, x');
      if f(x') > f(x*) then x* = x';
      if ub then update(p, x*, x, psize)
      else update(p, x', x, psize);
      i := i + 1;
    until (i > gens);
    return x*;
  end
```

(b)

```
Input: $N$ , $\chi$, $c_1$, $c_2$, $x_{\min}$, $x_{\max}$ (lower & upper bounds), $F$ (objective function).
Set $t = 0$.
Initialize $x_i^{(t)}, v_i^{(t)} \in [x_{\min}, x_{\max}]$, $p_i^{(t)} \leftarrow x_i^{(t)}$, $i = 1, \ldots, N$.
Evaluate $F(x_i^{(t)})$.
Determine the indices $g_i$, $i = 1, \ldots, N$.
While (stopping criterion is not satisfied) Do
    Update the velocities $v_i^{(t+1)}$, $i = 1, \ldots, N$, according to (1).
    Set $x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$, $i = 1, \ldots, N$.
    Constrain each particle $x_i$ in $[x_{\min}, x_{\max}]$.
    Evaluate $F(x_i^{(t+1)})$, $i = 1, \ldots, N$.
    If $F(x_i^{(t+1)}) < F(p_i^{(t)})$ Then $p_i^{(t+1)} \leftarrow x_i^{(t+1)}$.
    Else $p_i^{(t+1)} \leftarrow p_i^{(t)}$.
    Update the indices $g_i$.
    When (local search is applied) Do
        Choose (according to one of the Schemata 1–3) $p_q^{(t+1)}$, $q \in \{1, \ldots, N\}$.
        Apply local search on $p_q^{(t+1)}$ and obtain a new solution, $y$.
        If $F(y) < F(p_q^{(t+1)})$ Then $p_q^{(t+1)} \leftarrow y$.
    End When
    Set $t = t + 1$.
End While
```

(c)

```
Procedure daemon_actions
    for each $S_k$ do local_search($S_k$) {optional}
    rank $(S_1, \ldots, S_m)$ in decreasing order of solution
    quality into $(S_1', \ldots, S_m')$
    if (best($S_1'$, $S_{global-best}$))
        $S_{global-best} = S_1'$
    end if
    for $\mu = 1$ to $(\sigma - 1)$ do
        for each edge $a_{rs} \in S_\mu'$ do
                $\tau_{rs} = \tau_{rs} + (\sigma - \mu) \cdot f(C(S_\mu'))$
        end for
    end for
    for each edge $a_{rs} \in S_{global-best}$ do
        $\tau_{rs} = \tau_{rs} + \sigma \cdot f(C(S_{global-best}))$
    end for
end Procedure
```

(d)

```
Procedure ACO_Metaheuristic
    parameter_initialization
    while (termination_criterion_not_satisfied)
        schedule_activities
                ants_generation_and_activity()
                pheromone_evaporation()
                daemon_actions() {optional}
        end schedule_activities
    end while
end Procedure


Procedure ants_generation_and_activity()
   repeat in parallel for k=1 to m (number_of_ants)
      new_ant(k)
   end repeat in parallel
end Procedure


Procedure new_ant(ant_id)
    initialize_ant(ant_id)
    L = update_ant_memory()
    while (current_state $\neq$ target_state)
        P = compute_transition_probabilities(A,L,$\Omega$)
        next_state = apply_ant_decision_policy(P,$\Omega$)
        move_to_next_state(next_state)
        if (on_line_step-by-step_pheromone_update)
            deposit_pheromone_on_the_visited_edge()
        end if
    L = update_internal_state()
    end while
    if (online_delayed_pheromone_update)
        for each visited edge
            deposit_pheromone_on_the_visited_edge()
        end for
    end if
    release_ant_resources(ant_id)
end Procedure
```

(e)

Initialize pop_size antibodies
For each p in initialial population, p = local search (p)
Select m best antibodies from initial population
Proliferate the m antibodies, copy them to clonal library
$i = 1$
Select p_1, p_2 from clonal library
r = recombination (p_1, p_2), then r = local search (r)
Add r to the clonal library
$i = i + 1$
N
i = cr?
Y
$i = 1$
Select p from clonal library
t = mutation (p), then t = local search (t)
Add t to the clonal library
$i = i + 1$
N
i = cm?
Y
Replace s worst antibodies in initial population by the best antibodies in the clonal library
Terminated ?
N
Y
Output

(f)

Fig. 1 In (a) the pseudocode (reproduced from [4]) for a Memetic Learning Classifier System, (b) the pseudocode (reproduced from [96]) for an Estimation of Distribution-Like Memetic Algorithm, (c) a Memetic Particle Swarm Optimisation pseudocode (reproduced from [115]), (d) & (e) pseudocode of a representative Ant Colony Optimisation metaheuristic (reproduced from [31]) with a solution refining strategy, through local search, for ACO and (f) an AIS inspired Memetic Algorithm flowchart (reproduced from [142])

use of the Template Pattern one can have a very generic and reusable recipe for implementing solutions to a range of, perhaps very different, problems.

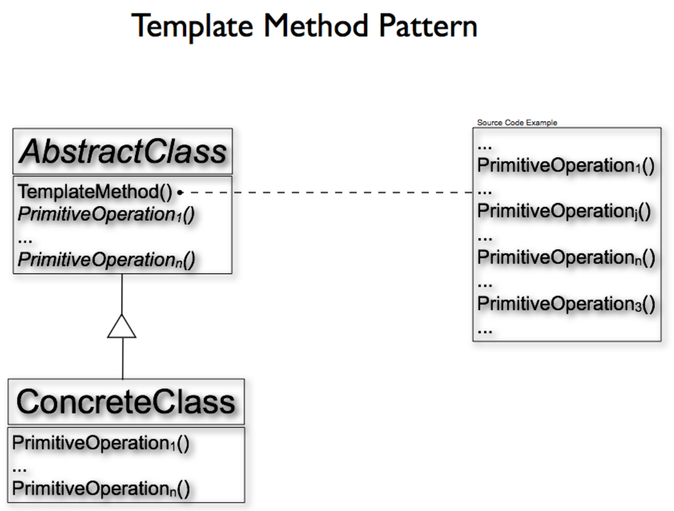
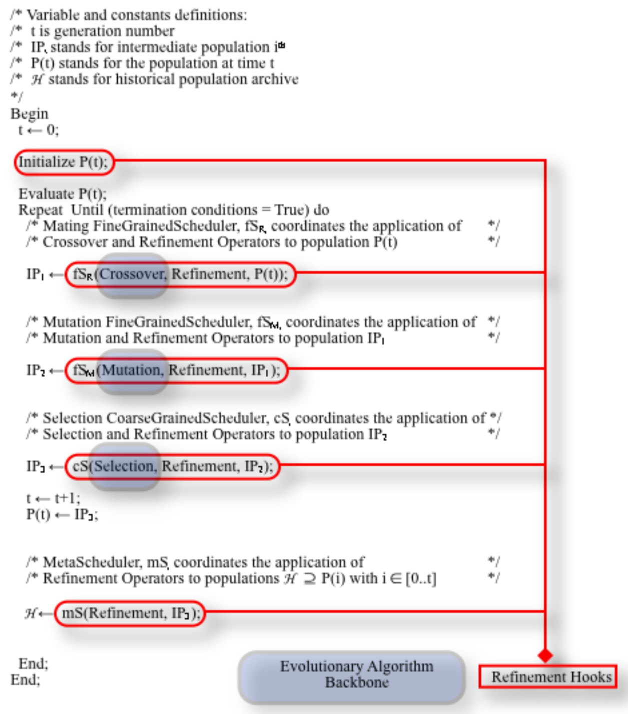


Fig. 2 A UML class diagram sketching the structure for implementing Template Method patterns (adapted from [43])

Figure 2 shows a class diagram capturing a Template Method pattern. The abstract class defines a template method that provides the algorithmic skeleton for a specific functionality. To achieve its functionality the template method calls one or more primitive operations (defined abstractly in the abstract class) that can be redifined in more specialised subclasses (concreteClass in the figure). Figures 1(d) and 1(e), 1(f), 1(c), 1(b) and 1(a) are examples of Template Method Patterns for ACO, AIS, PSO, EDA and LCS based Memetic Algorithms respectively. Each TP captures the invariant properties of Memetic Algorithms when it is implemented from the perspective of a specific nature-inspired algorithms. It also captures the invariant features of MAs regardless of which nature-inspired route is used to implement it, e.g., the entwining of global and local search procedures. In what follows we will discuss other design patterns that are critical to the implementation of competent Memetic Algorithms thus extending the pattern language for MAs.

2.1 A Template Pattern for an Evolutionary Memetic Algorithm

As the design issues and critical considerations behind the implementation of MAs are almost identical across the various possible nature-inspired algorithms that could be used as templates and, on the other hand, as Evolutionary Memetic Algorithms are the best known MAs, we will focus the following discussions on an Evolutionary Algorithm Template Pattern for MAs. The pattern language that will emerge, however, will be valid for other nature-inspired paradigms as well.



```
/* Variable and constants definitions:
/*  t is generation number
/*  IP_i stands for intermediate population i^th
/*  P(t) stands for the population at time t
/*  𝓗 stands for historical population archive
*/
Begin
  t ← 0;
 Initialize P(t);
 Evaluate P(t);
 Repeat  Until (termination conditions = True) do
   /* Mating FineGrainedScheduler, fS_R, coordinates the application of      */
   /* Crossover and Refinement Operators to population P(t)                  */
   IP_1 ← fS_R(Crossover, Refinement, P(t));
   /* Mutation FineGrainedScheduler, fS_M, coordinates the application of    */
   /* Mutation and Refinement Operators to population IP_1                   */
   IP_2 ← fS_M(Mutation, Refinement, IP_1);
   /* Selection CoarseGrainedScheduler, cS, coordinates the application of */
   /* Selection and Refinement Operators to population IP_2                  */
   IP_3 ← cS(Selection, Refinement, IP_2);
   t ← t+1;
   P(t) ← IP_3;

   /* MetaScheduler, mS, coordinates the application of                      */
   /* Refinement Operators to populations 𝓗 ⊇ P(i) with i ∈ [0..t]           */
   𝓗 ← mS(Refinement, IP_3);
 End;
End;
```

Fig. 3 An Evolutionary Algorithm based Memetic Algorithm Template. The figure highlights the EAs core operators as well as the hotspots where the algorithm can be refined, i.e., where "memetic" operators might come into play

NAME: Evolutionary Memetic Algorithm Template Pattern (EMATP)[2]

- *Problem statement:* the EMATP provides a viable route for solving the problem of how to best coordinate global and local search methods from within an evolutionary algorithms paradigm. Although in principle the simultaneous exploration of the search space by the evolutionary process and the exploitation (by refinement) of candidate solutions should result on an improved algorithm, this is not always the case. The expectation is that a hybridisation of an EA would result in a synergistic net effect that would productively balance local and global search.
- *The solution:* the (almost) standard evolutionary cycle composed of $Initiate \rightarrow Evaluate \rightarrow Mate \rightarrow Mutate \rightarrow Select \rightarrow RepeatUntilDone$ is expanded with domain-specific operators that can refine this cycle. The refinement processes could vary for the different elements of the core EA's pipeline and could be implemented through exact, approximated or heuristic (e.g. local search) methods. Domain specific operations are often implemented in the form of smart initialisations, local search procedures, etc that refine the input solutions to the Mate and/or Mutate processes as well as (more often) their outputs. Refinements could also be applied to the selection, initialisation and population management processes through e.g. fitness sharing, crowding, population structuring (e.g. cellular/lattice structures, demes, islands, etc) and diversity management strategies.
- *The consequences:* paradoxically, although the key motivation for using Memetic Algorithms has been to refine solutions and converge towards good local optima (or ideally global optima) fast, sometimes the balance of local and global search is poorly implemented resulting on an untimely diversity crisis because the algorithm converges too fast. Another direct consequence of using Memetic Algorithms is that often, refining solutions by problem specific strategies incurs in a substantial additional CPU commitment. Thus it becomes essential to validate whether the exploration mechanisms of the core EA's pipeline when augmented with refining strategies do not end up producing worse solutions than having run the pure EAs with the same total CPU commitment (or the refinement strategies alone with an equivalent total CPU budget).
- *Examples:* figure 3 shows a concrete example of an Evolutionary Memetic Algorithm. A detailed description and analysis of several successful implementations of the EMATP pattern are described in [82]. The paper also provides a taxonomy for EMAs as well as a discussion of design issues.

Figure 3 shows a concrete example of the EMATP. In the figure we have identified the key EA's components as the "backbone" and the potential places where refinement might take place as refinement hooks. Each hook is represented by a "scheduler" operator that manages the flow of information (e.g. partial/candidate solutions, time-dependent parameters, etc) between each of the core EA's component and the refinement strategies associated with them. A formal notation for these schedulers can be found in [82]. A detailed study of the literature reveals that three types of schedulers can be abstracted from the various templates that are implemented. These schedulers are: a *fine-grain scheduler* for coordinating the operation of the genetic operators mutation and crossover with refinement methods, a *coarse-grain scheduler* for overseeing the interplay between population management strategies (e.g. selection) and refinement strategies and *meta scheduler* that orchestrate refinement procedures over longer timescales and larger spatial scales (e.g. islands, population structures, pareto archives, etc). These schedulers receive their names because they operate at different level of granularities and timeframes. The fine-grain schedulers (represented by fS_R, fS_M in Fig. 3) have access to a very limited number of solutions and hence they have a very localised view of the current state of the search and thus the decision they can take in terms of parameter and operators adaptation is confined to small regions of the search space. The course-grain scheduler (represented by *cS* in Fig. 3), on the other hand, has access to a complete population and, perhaps, even to intermediate populations that could have been created in the main EA's pipeline. Thus it have a more global view of the search state that might be used to strategically guide the application of further refining methods to (parts of) the population it has access to. The meta-scheduler (represented by *mS* in Fig. 3), on the other hand, has access to potentially the power set of all previously visited solutions and hence it has an even broader informational base upon which to decide the appropriate balance between exploration and exploitation. These schedulers also operate at different time-scales. Clearly, fS_* can operate very frequently indeed, potentially each time that a crossover or mutation event takes place. The coarse-grain scheduler operates once per generation and the *mS*, potentially, even less often. These differing spatial (i.e. access to solutions) and temporal (frequency of operation) features result in constraints on the complexity of the algorithmic strategies that these schedulers can implement and have a direct impact in several design issues such as whether or not a surrogate fitness function [144, 145] should be used, etc. Needless to say, EMATP can be implemented through a series of (object-oriented) class hierarchies. The EMATP pattern in Fig. 3 can be encapsulated into an abstract class from which subclasses implement specific versions of the pattern. One could thus imagine a family of Evolutionary Memetic Algorithms where one or more features are either removed from the pattern or added to it simply by overriding the behaviour of the scheduler methods.

2.2 Strategy Patterns for Memetic Algorithms Design Issues

We extend the Memetic Algorithms pattern language by describing a series of Strategy Patterns associated with design issues arising from attempting to implement effective instances of the EMATP. The Evolutionary Algorithm's processes such as crossover and mutation, as well as the refinement strategies and the schedulers that coordinate their operations, are best thought of as a Strategy Patterns [43]. These patterns are useful for defining a family of interchangeable algorithms.

The idea is that the client that employs a member of this family of algorithms, e.g. the fine-grained scheduler coordinating a local search strategy with a crossover operator, should be able to use any of the available local search and/or crossover strategies without need for recoding. A most frequent example is provided when testing different, e.g., crossover operators such as one-point crossover, two-point crossover and uniform crossover without the need to rewrite the EA's backbone.

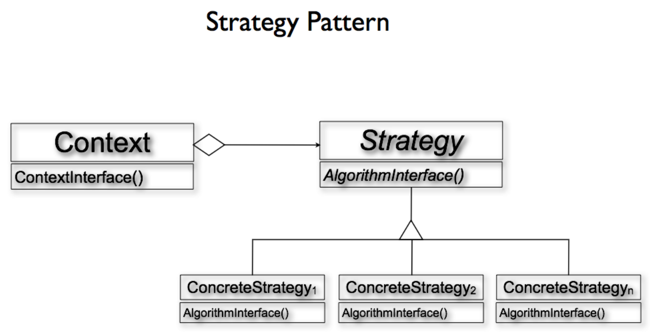


Fig. 4 A UML class diagram sketching the structure for implementing Strategy Patterns (adapted from [43])

Similarly, one should be able to change the local search used without affecting the pattern in which it is used. That is, a Strategy Pattern should be used each time that a program, in this case a Memetic Algorithm, needs a specific service or function and when there are, in fact, several ways of executing these functions. To promote reusability, Strategy Patterns follow a scheme similar to the one depicted in Figure 4. The choice of which strategy to implement as to perform the required function is problem and instance specific. Thus by using Strategy Patterns, one can change algorithms without causing a chain-reaction of changes in the code that uses those algorithms. As we will argue below, there is a direct link between critical Memetic Algorithms design issues and Strategy Patterns.

2.2.1 Refinement Strategies

We have argued that an EMATP provides conceptual solutions to the problem of balancing local and global search. The global search is performed through the implementation of some form of Evolutionary Algorithm while the local search is meant to be implemented by some form of refinement operator. The choice of operator refinement is a critical design decision in Memetic Algorithms engineering as it impacts tremendously the synergy between the EA's backbone and the refinement methods of choice.

NAME: Refinement Strategy Pattern (RSP)[3]

- *Problem statement:* this pattern attempts to address the following key questions: (1) what refinement operator should be used and ultimately, (2) what fitness landscape will the MA be exploring? While the EMATP helps define the algorithmic skeleton of an Evolutionary-based Memetic Algorithm, it leaves to the Strategy Patterns the problem of how to decide what strategy would be implemented at the various refinement stages. Indeed, both theoretical and experimental work has analysed the effectiveness of different local search strategies within Memetic Algorithms.
- *The solution:* the key design issue related to refinement strategies is how they prevent getting trapped in (poor) local optima, i.e. search stagnation, while performing an efficient and fast local search. The answer to questions 1 and 2 above is problem, instance and time dependent, hence a solution can only be provided at an abstract engineering level that allows for the seamless adaptation of the EMATP to different situations. More precisely, for a given set of low-level move operators, e.g. n-exchange, n-swaps, bit-flip, etc, a variety of navigation rules have been implemented. Navigation rules specify how the search landscape induced by a given move operator is traversed. Some of the alternatives are breadth first search, depth first search, variable depth, etc. Besides the navigation method, the refinement strategy need also specify an acceptance criteria, e.g. greedy, Monte Carlo, Great-Deluge[36, 15], etc. Finally, the most sophisticated refinement strategies employ multiple move operators and acceptance criteria, e.g. Multimeme algorithms [72, 81], variable-neighborhood search [48, 49], etc. Indeed, a refinement strategy pattern might use at its core a full Memetic Algorithm Pattern, see [122] for an example. Thus through the recursive nesting of MAP and RSP it is possible to obtain extremely complex and versatile search methodologies.
- *The consequences:* the Refinement Strategy Pattern comes with some "strings attached" to it. Its very flexibility means that a given Memetic Algorithm might be exploring not one but several search landscapes simultaneously. The precise nature of this combined search space should, ideally, be adequately analysed either experimentally or theoretically (ideally in both ways). Fitness landscapes statistics were utilised in, e.g., [66, 97]. Krasnogor and Smith in [83] resorted to Polynomial Local Search Theory to show the impact of various combinations of local search and genetic operators. The paper provides a worst case analysing for the complexity of heuristics applied to a well-known problem, the two-dimensional and Euclidean TSP. By way of simple arguments the paper shows the PLS-completeness of a family of memetic algorithms for the TSP as well as graph partitioning and maximum network flow (both with important practical applications). It was shown that potentially very long paths to local optima exists even when the neighbourhood used by some MAs are of polynomial size. Interestingly, the paper's arguments are also valid for a variable-neighborhood search, rather than evolutionary based, template pattern for global search. A similar result is also provided in [132] for MinCut, MaxSAT and Knapsack problems.
- *Examples:* intelligent Refinement Strategy Patterns can be seen in [60, 105]

2.2.2 Exact and Approximate Hybridisation

In the previous section we have presented a Refinement Strategy Pattern that dealt with the general question of how to design a navigable fitness landscape for Memetic Algorithms through a judicious selection of refinement strategies. One complementary and related design issue is that of the hybridisation of Evolutionary Algorithms with exact and approximate methods for which we present a new strategy pattern.

NAME: Exact and Approximate Hybridization Strategy Pattern (EAHSP)[4]

- *Problem statement:* The integration of exact and approximation method into an EMA pattern requires the consideration of specific design issues that are distinct than those of other (e.g. heuristic) refinement strategies. More specifically, the development of exact methods requires considerably more effort than that of a set of relatively simple local searchers. The majority of the effort goes into obtaining a tight formulation of the problem for the exact method or a suitable relaxation. Moreover, exact and approximation methods might take a substantial amount of CPU effort to run, sometimes dwarfing that required by the EMA. Thus how to synergistically exploit both approaches for obtaining a better solver remains a critical problem.
- *The solution:* It has been argued that there are essentially two main ways of integrating exact and EMA methods. The first approach is based on running the EMA and the exact method in tandem, that is, the EMA provides solutions to the exact method (e.g. a Branch and Bound method) that could, in turn, use the evolved solutions as good bounds thus helping reduce the Branch and Bound runtime. Symmetrically, the exact method might be used to seed EMA populations with good candidate solutions. In this loose integration each algorithm runs essentially independent from each other but exchanging information from time to time. On the second implementation strategy the Branch and Bound calls the EMA from time to time and thus uses the EMA as an heuristic to produce bounds. It might call it only once as to provide initial solutions or many times over its execution thus forming a more tightly coupled pipeline. [42] calls these two strategies coercive and cooperative respectively. Sometimes, when the problem is excesively large, e.g. it does not fit in memory and there is no possibility of a distributed computation, Branch and Bound is heuristically modified and converted into a Beam Search. Although this approximation loses peformance guarantees it does allow for much larger problems to be tackled[42]. In [116] the authors interleave the execution of an EMA with a Lagrangian relaxation method based on a loose coupling strategy.
- *The consequences:* the application of this strategy pattern requires a detailed modeling of the problem at hand and a realistic evaluation of the exact method and Memetic Algorithm relative runtimes. It should be clear that this strategy should be used if the added benefit of guaranteed results outweigh the additional efforts both in engineering the system and in running it.

- *Examples:* [98] explores the application of both a tandem and pipeline strategies for the bi-objective flowshop scheduling problem. [119] provides a detailed analysis of how to combine integer and linear programming methods with metaheuristics. Many of the strategies reported are directly applicable to EMAP.

2.2.3 Population Diversity Handling Strategies

A key characteristic of Memetic Algorithms is that they combine, on the one hand, search through a population of solutions with, on the other hand, search focused around specific promising ones. The introduction of refinement strategies to improve over a subset of solutions often produces additional selection that, in turn, brings a premature population diversity crisis. A population's lack of diversity causes the MA to allocate repeated reproductive, mutation and refinement trials to the same individuals thus wasting precious CPU resources. Balancing population diversity is a critical issue, that in itself necessitates a detailed analysis of what it is meant by "diversity". The complexity of this task is represented in Figure 5. Only in the most idealised problem the mapping from genotypes to phenotypes and then to fitness values is a one to one direct mapping. In real-world applications these mappings are highly complex. In some cases, the interpretation of a given solution representation (i.e. genotype) into a phenotype (i.e. actual solution) is nonlinear and/or stochastic and results in potentially different phenotypes for the same genotype. Similarly, fitness assignment to phenotypes might also be nonlinear and/or stochastic thus producing, for a given phenotype,potentially different fitness values. The existence of nonlinearities and stochasticity in the genotype $\rightarrow$ phenotype $\rightarrow$ fitness mappings gets compounded when multi-objective problems are tackled. Hence, the definition of population diversity used by diversity strategies must take these complex mappings into account.

NAME: Population Diversity Handling Strategy Pattern (PDHSP)[5]

- *Problem statement:* the goal of this pattern is to provide interchangeable strategies for the robust management of population diversity within Memetic Algorithms.
- *The solution:* sustaining population diversity in Memetic Algorithms is a difficult task. It is usually tackled by a combination of strategies operating at various levels. First, the population is usually initialised in an intelligent manner as to avoid re-sampling and non-representative coverage of the search space at the beginning of the search. Examples of initialisation mechanisms can be found in [24, 84]. Later, during the search, population diversity maintenance can be implemented at various levels. For example [74] uses a memory of solution features during the mating process as to avoid generating phenotypes with over-represented features. In this work population diversity is controlled at the phenotype level. An example of a diversity preservation strategy at the fitness level is adopted by [69] where the selection of which local search to use is guided by the range of fitness in the population. Similarly, [80] uses an adaptation in the local searcher to monitor

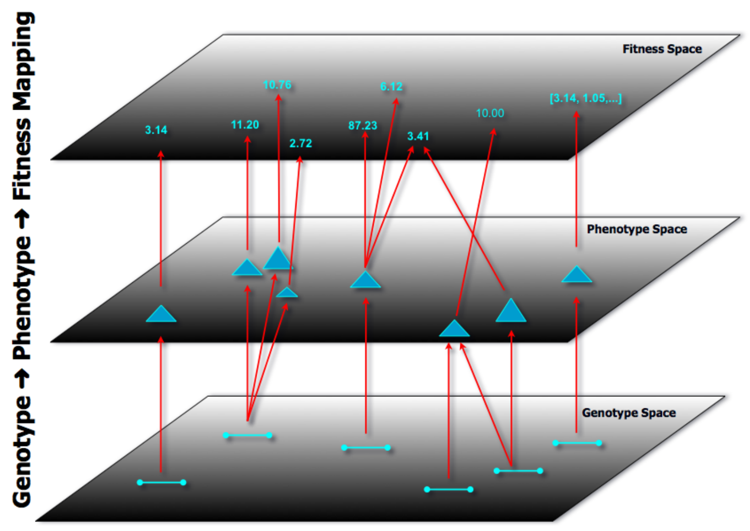


Fig. 5 The mapping from genotype (i.e. solution encoding) to phenotype (i.e. encoding interpretation) to fitness (i.e. a concrete measure of a solution's worth) can be very complex

and affect fitness distributions. In the context of AIS based templates for Memetic Algorithms, the aging operator eliminates the oldest candidate solutions from a population as to contribute to maintaining diversity and thus avoiding (poor) local optima. AIS and Differential Evolution (DE) based solvers for multi-dimensional problems where Diversity Handling and Refinement Strategies are compared are reported in [32]. Keeping track of the "age" of solutions is quite an ubiquitous strategy used on several multi-solution templates. Sorensen and Sevaux [131] propose several population management strategies based on measuring diversity in the solutions space for multidimensional knapsack problem and weighted tardiness single-machine scheduling problems. For continous problems, diversity is sometimes preserved through structuring the population using a cellular memetic algorithm [107].

- *The consequences:* Care should be taken in designing the diversity measure to use. In [16, 17] we have analysed a variety of diversity measures for problems where mappings such as those depicted in Figure 5 takes place. Although the examples used in these papers are based on Genetic Programming, the lessons learnt are universal for Memetic Algorithms, namely, (1) increased diversity does not always positively correlate with improved performance and (2) whether it leads to improved performance depends on the complex mapping between genotypes, phenotypes and fitness. Thus population diversity handling strategies must be implemented in such a way that they should be easy to change and benchmark as to continuously assess their performance in specific problems.

- *Examples:* Neri et al. present a comparative study of several fitness diversity based adaptation schemes for Multimeme algorithms in [106] . Landa Silva and Burke[87] address the issue of the interplay between diversity and multi-objective optimisation.

2.2.4 Surrogate Strategies

Evolutionary Memetic Algorithms are usually used to solve very complex and hard problems. Oftentimes, these problems also give rise to uncertainties in the assignment of the fitness values of individuals upon which selection can be applied. On the other hand, refinement strategies used within Memetic Algorithms are usually computationally expensive, there is no explicit knowledge of a fitness function (e.g. in interactive evolutionary design problems) and hence fitness must be "reverse-engineered". Moreover, in some cases the objective function is extremely multi-modal and a smoothing criteria is required. The above considerations lead directly to an important design issue in Memetic Algorithms engineering, namely, the use of surrogate objective functions.

NAME: Surrogate Objective Function Strategy Pattern (SOFSP)[6]

- *Problem statement:* Memetic Algorithms are computationally-intensive methods and one – but not the only – of the most important computational bottlenecks that must be considered when engineering an MA is the calculation of the objective (i.e. fitness) function. As shown in Fig. 5, many problems are inherently noisy, i.e., the same candidate solution might be assigned a different fitness value due to stochasticity or non-linearities in their evaluation. In other cases, several promising solutions might be repeatedly improved by a Refinement Strategy Pattern and this refinement – almost invariably – leads to a large number of additional objective functions evaluations. In optimal design problems, one often finds that no explicit knowledge of fitness is available and the quality of a solution must be inferred from samples given by experts. The aim of the Surrogate Objective Function Strategy Pattern is to provide effective ways of replacing an expensive, noisy or unknown fitness function with a suitably defined approximation. The goal is to ameliorate the cost of highly expensive fitness function evaluations while preserving fitness assignment quality by providing a "proxy" to the original objective function that is faster than the former but of sufficient quality so that it can provide good quality estimations of the true fitness value of a candidate solution.
- *The solution:* Surrogate fitness functions have also been called metamodels, local models and partial objective functions in the literature. Although the use of effective strategies for solving the surrogacy problem are also important to other branches of natural computation, it is particularly poignant in Memetic Algorithms design. This is so because MAs should be able to cope with uncertainty in the genotype $\rightarrow$ phenotype $\rightarrow$ fitness mappings, which could lead to a noisy fitness assignment, ambiguity in the very definition of fitness but also MAs must

contend with and balance the additional computational effort introduced by the refinement strategies. Explicit averaging has been used to ascertain the fitness of candidate solutions under uncertainty. This is exemplified by reduction of variance techniques (e.g. Latin Hypercube Sampling), re-evaluation of distinct individuals (e.g. those with high fitness or variance), weighted histories [10, 11, 12], etc. Fitness inheritance [92] Artificial Neural Networks and related approaches [112], Models[13] and Metamodels [8] as well as Design of Experiments [123] have also been used for surrogate fitness implementation. In [62, 113] the author presents a detailed analysis of surrogate methods for evolutionary computation.

- *The consequences:* The use of a Surrogate Objective Function Strategy is, in many real-world scenarios, unavoidable. However, its use brings forth a series of other design decisions the Memetic Algorithms engineer will need to take into account, e.g., at what level will the approximation be more productive? the fitness level or the problem itself? is it possible to use global models or should a collection of local approximations be preferred, etc.
- *Examples:* [145] provides specific recommendations for integrating approximate fitness models within Memetic Algorithms.

2.2.5 Continuous Problems

The utilisation of EMATP for continuous problems presents, in addition to the previous design considerations, new opportunities and challenges. In this section we will only focus on the issue of integrating refinement strategies for continuous optimisation within a Memetic Algorithm framework. The reader must note that there is a large body of literature on the use of evolutionary computation for continuous domains, which we do not tackle here. Unlike combinatorial optimisation where it is always possible to detect when a given point is a local optimum, in the continuous case it is, in general, not possible and hence one must settle for an a priori decision on what solution precisions are acceptable or provide a decision making mechanism to the MA itself. This becomes even more critical when facing continuous design optimisation problems with multiple optimality criteria (e.g. [117]). In what follows we describe for continuous problems a Refinement Strategy Pattern that addresses some of the design issues reported in [52] .

NAME: Continuous Problems Refinement Strategy Pattern (CPRSP)[7]

- *Problem statement:* Effective search requires an appropriate determination of search scales, which in continuous optimisation is not always possible, e.g., it is often impossible to determine, given a candidate solution, whether it represents a local optimum or not. The lack of explicit local scale might also result on very long searches that when combined with lack of gradient information pose a very difficult problem for optimisation techniques. Thus, whether one uses as a refinement strategy a derivative-free method such as, e.g., Nelder-Mead Simplex[104] or Lagrangian Interpolation [7], or methods that use first or second

order information, e.g., Gram-Schmidt orthogonalization or Broyden-Fletcher-Goldfarb-Shanno [64], [57], in general, it is not advisable to rely on detecting a local optimum to stop a local search within a continuous problem.
- *The solution:* Schwefel in [124] surveys several continuous optimisation methods some of which use derivatives information and some which do not. Any of these methods can be integrated within an EMATP through a judicious balance of the amount of effort allocated to the local searchers. As in general it is not possible to run these standard methods all the way to stationary points or local optima one relies on either truncated searches and a selective application of the local continuous optimisation technique to some of the potential solutions in the population. Local search frequency and intensity parameters defining the proportion of individuals in the population that will undergo refinement and how much effort each one will be allocated are introduced. As mentioned in [107] the values for these parameters and the decision of which of the above mentioned refinement methods to use is very much problem dependent. As it is the case for discrete optimisation problems, MAs applied to continuous domains also, if improperly engineered, might suffer from premature convergence through a rapid loss of diversity in the population. The use of infrequent or incomplete searchers contributes to maintaining diversity but it has been shown, e.g. [118], that a lattice-type population structure promotes a more diverse search.
- *The consequences:* The difficulty of a priori deciding step sizes, local search probability, refinement strategy, etc for each new problem suggests that, unless one knows before hand the problem's characteristic features, an adaptive mechanism must be used to on-the-fly decide on the various choices available. Similarly to the combinatorial case[74], Ong and Keane propose to use Multimeme algorithms for continuous optimisation problems [110].
- *Examples:* Other examples on intelligent hibridisation strategies for continous problems could be seen in [93, 100]. In [50, 51] it is argued that many of the difficulties in guaranteeing an efficient combination of local-global search resulting in adequate convergence properties might be tackled by discretising the decision space. These papers provide both theoretical as well as practical guidance on how to achieve this.

2.2.6 Multimeme Strategy Pattern

In previous sections we have argued that, for the Evolutionary Memetic Algorithm Template pattern to be of practical use, it should be further decorated with a series of Strategy patterns. Most of these Strategy patterns are conceptually linked to considerations about MA's design issues. Chief amongst these issues is the selection for a given problem instance, at a given point in time during the search process and for a given potential solution (out of a potentially large population), which Refinement Strategy to use. Deciding which Refinement strategy to use can be further extended to decide not only amongst heuristic methods but also amongst approximate and exact methods in both the discrete and continuous case. The following Strategy pattern addresses this design issue.

NAME: Multimeme Strategy Pattern (MSP)[8]

- *Problem statement:* Deciding which type of refinement or complementary search strategy to use at any given time for a given problem instance and a set of candidate solutions is a far from trivial matter. This is particularly difficult in the absence of abundant statistical information on an algorithm's behavior in relation to problem instances' features.
- *The solution:* Rather than deciding a priori on a particular search strategy (meme) by which to complement an EMATP, one can incorporate adaptive or self-adaptive techniques that would allow the MA to dynamically change the refinement strategy accordingly to how search is progressing. These methods have been called Multimeme Algorithms. Several relatively simple schemes have been shown to be extremely successful both for discrete and continuous optimisation. For example [81, 74, 79, 26, 72] assign to each available local search method-meme a label which, during crossover, is inherited by an offspring from the fittest parent. Thus, the most effective local search methods are adaptively used during the search process and the fittest problem solving memes survive over generations. As to maintain local search diversity, the mutation operator can also override an inherited local search label and assign a new one, thus ensuring that search is also done in the heuristic (memetic) space. Similarly, for continuous optimisation [110] presents Multimeme algorithms that employ on-line reinforcement performance information to adaptively change the memes used. Also, a roulette wheel decision mechanism can be exploited for sampling memetic space, i.e. stochastically selecting (with bias) from the set of available local searchers the one to use next. An extensive discussion on how to "schedule" various local searchers-memes within an EMATP, and the levels at which search performance information can be incorporated into the decision making process behind the selection of each refinement strategy, is available in [82, 83].
- *The consequences:* The on-line adaptation of the local search choice requires some kind of bookkeeping mechanism as to ascertain the usefulness of the various refinement memes during the search. Besides the simple mechanisms for scheduling Multimeme algorithms appearing in the references mentioned above, [129] discusses other ways of assigning credit to a potentially varying set of local searchers. In principle, any reinforcement-learning [65] type of mechanism could be used. A related family of algorithms, called Hyperheuristics [22, 67], has been proposed as an attempt to raise the level of generality in problem-solving by intelligently managing the application of collections of "low level" heuristics. The learning mechanisms behind Hyper-heuristics could also be used within Multimeme algorithms (and viceversa).
- *Examples:* Other examples of Multimeme strategies appear in [60, 105]. Multimeme strategies are closely related to adaptive and self-adaptive methods hence the reader should also refer to the vast literature on the subject. Some pointers are [130, 125, 129, 86]

2.2.7 Self-Generating Strategies

Before describing a Strategy pattern for Self-Generating Strategies, we revisit the etimology of "memetic" in Memetic Algorithms and we argue that, indeed, there is a promising nature-inspired research direction that could lead to important new algorithmic variants. Memetic theory started as such with the introduction by R. Dawkins of the concept of a meme in [33] and later refined in [34]:

> I think that a new kind of replicator has recently emerged on this very planet. It is staring us in the face. It is still in its infancy, still drifting clumsily about in its primeval soup, but already it is achieving evolutionary change at a rate that leaves the old gene panting far behind. The new soup is the soup of human culture. We need a name for the new replicator, a noun that conveys the idea of a unit of cultural transmission, or a unit of imitation. "Mimeme" comes from a suitable Greek root, but I want a monosyllable that sounds a bit like "gene". I hope my classicist friends will forgive me if I abbreviate mimeme to meme.(2) If it is any consolation, it could alternatively be thought of as being related to "memory", or to the French word "même". It should be pronounced to rhyme with "cream". 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 sperms 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.

The fact that cultural phenomena could, in certain cases, be modeled or understood as some sort of evolutionary process was not a new idea, .e.g., the elementary unit of cultural change and/or transmission was sometimes called *m-culture* and *i-culture* [29], *culture-type* [121], etc. Durham [38] provides a quite compelling case for the co-evolution of genes and culture in H.Sapiens. The fundamental innovation of memetic theory is the recognition that a dual system of inheritance, by means of the existence of two distinct replicators, moulds human culture. Moreover, these two replicators interact and co-evolve shaping each other's environment. As a consequence evolutionary changes at the gene level are expected to influence the second replicator, the memes. Symmetrically, evolutionary changes in the meme pool can have consequences for the genes. From a computer science perspective, this is a very appealing nature-inspired computation paradigm as it defines a possible two-scale learning and adaptation mechanism, one at the level of solutions to problems in exactly the same way in which EAs (or any other population based metaheuristic) attempts to solve problems and a second level in which the solving methods themselve evolve, or more generally, change. The distinction between standard EAs and a Meme-Gene Evolutionary process can be seen in Figures 6(a) and 6(b) respectively.

In Figure 6(a) a hypothetical population of individuals is represented at two different points in time, generation 1 (G1) and at a later generation (G2). In the lower line, G_i for $i = 1,2$ represents the distribution of genotypes in the population. In the upper line, P_i represents the distribution of phenotypes at the given time. Transformations T_A account for epigenetic phenomena, e.g., interactions with the environment, in-migration and out-migrations, individual development, etc., all of them affecting the distribution of phenotypes and producing a change in the distribution of genotypes during this generation. On the other hand transformations T_B account for the Mendelian principles that govern genetic inheritance and transforms a dis-

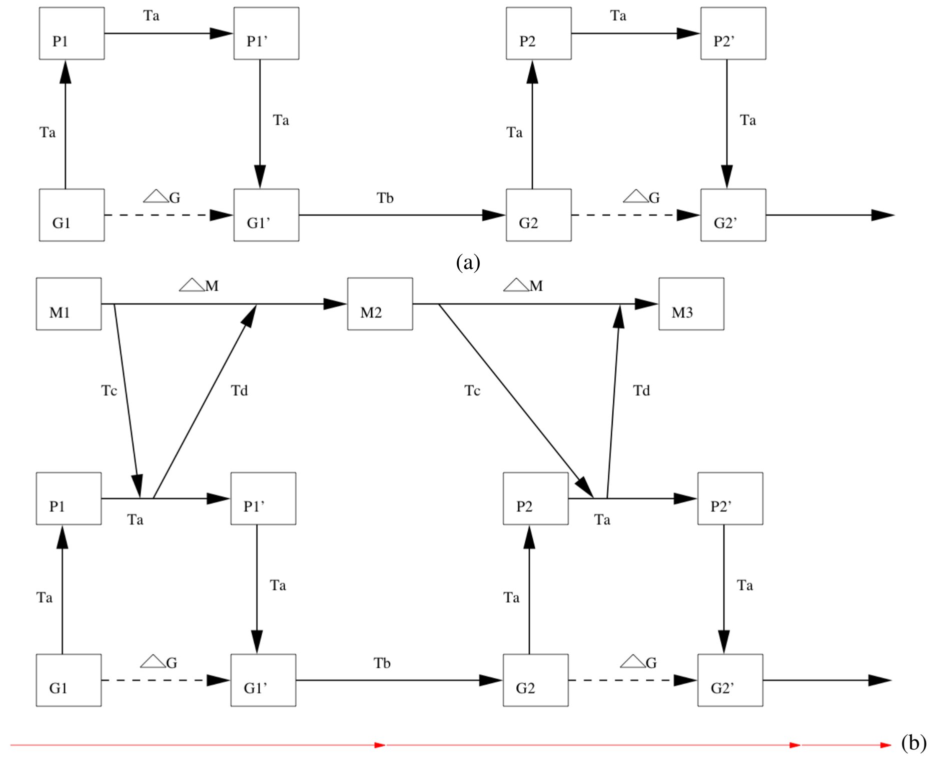


Fig. 6 In (a) evolutionary genetic cycle (adapted from [38] page 114), (b) Coevolutionary memetic-genetic cycle (adapted from [38] page 186)

tribution of genotypes G_1' into another one G_2. Evolutionary computation endeavors concentrate on the study and assessment of many different ways the cycle depicted in 6(a) can be implemented. This evolutionary cycle implicitly assumes the existence of only one replicator, namely genes, representing solutions to hard and complex problems. On the other hand, Figure 6(b) reflects a coevolutionary system where two replicators of a different nature interact. In the context of memetic algorithms, memes represent instructions to self-improve. That is, memes specify sets of rules, programs, heuristics, strategies, behaviors, etc, defining the very essence of the underlying search algorithm. Feldman and Cavalli-Sforza[27] suggest that memetic-genetic search processes might be driven by:

> ...the balance of several evolutionary forces: (1) *mutation*, which is both purposive (innovation) and random (copy error); (2) *transmision*, which is not as inert as in biology [i.e., conveyance may also be horizontal and oblique]; (3) *cultural drift* (sampling fluctuations); (4) *cultural selection)* (decisions by individuals); and (5) *natural selection* (the consequences at the level of Darwinian fitness) ...

Figure 6(b) shows the type of transformations that are possible under a coevolutionary setting, namely, the usual transformations in terms of genes and phenotypes distributions but also meme-phenotypes and memes-memes processes. There are mainly two transformations for memes that are depicted, T_C and T_D. Transformations T_C represents the various ways in which "cultural" instructions can re-shape phenotypes distributions, e.g., individuals learn, adopt or imitate certain memes or modify other memes. T_D, on the other hand, reflects the changes in memetic distribution that can be expected from changes in phenotypic distributions, e.g., those attributed to teaching, preaching, etc. Thus Dawkin's memes, and their extensions, e.g. [27, 38, 41, 9], as evolvable search strategies [70, 71, 73, 77, 128, 19] provide a critical link to the possibility of open-ended combinatorial and/or continuous problem solving.We describe next a Self-Generating Strategy Pattern that captures some of the solutions for implementing the above concepts.

NAME: Self-Generating Strategy Pattern (SGSP)[9]

- *Problem statement:* The essential problem these strategies tackle can be succinctly put as "how to implement a search mechanism that learns how to search?" or "how to optimise through learning in a reusable manner?" [2]This problem statement is not about adaptation or self-adaptation mechanisms, that mainly deal with parameter learning within fixed algorithmic templates, but rather how to infer, on-the-fly, new algorithmic templates that could be useful at a later time.
- *The solution:* Needless to say, this is one of the more recent cutting-edge developments in MA and the proposed solutions are still in their infancy. Self-Generating MAs (sometimes called coevolutionary MAs) were first proposed as co-evolutionary MAs in [70] and these first ideas were explained in some more detail in [71]. The first experimental confirmation of the potential behind these strategies came with [75, 126, 127, 76, 77, 73]. The key concept behind these new algorithms was the capturing of algorithmic building blocks through a formal grammar. Through a process not unlike grammatical evolution[109], sentences in the language induced by the grammar were evolved. These evolved sentences represented entire search methods that were self-generated on-the-fly while attempting to solve a given problem. The above algorithms were used for a range of difficult problems such as polynomial time solvable with low epistasis NK landscapes, polynomial time solvable with high epistasis NK landscapes all the way up to NP hard NK landscapes with both low and high epistasis as well as in randomly generated benchmarks, simplified models of proteins structure prediction and on a real world bioinformatics application, namely, protein structure comparison.
- *The consequences:* It is evident that a Self-Generating strategy can only be used in a setting where the problem is either sufficiently hard that it is worth paying the additional cost of searching an algorithmic design space or for which no good heuristics are available and one is interested in evolving them. Interestingly, after one has evolved a set of refinement strategies these can be readily used (without need to re-discover them) in new instances or problems, under a

[2] Please note that we are not referring here to the well known fact that many learning processes can be recast as optimisation ones.

Multimeme Strategy setting. Thus the Self-Generating mechanisms described in the papers mentioned above provide a concrete implementation for the nature-inspired gene-meme coevolutionary process depicted in Figure 6(b).

- *Examples:* Recent papers exploiting GP type learning processes for evolving problem solvers are [44, 18, 19, 20, 114, 6, 5, 40, 133, 135]. It is important to note that from this set of examples, only [40] evolve algorithms, the others evolve rules or evaluation functions that guide other heuristics. Thus [75, 126, 127, 76, 77, 73, 40] are the closest examples available of a coupled optimisation-learning process such as the one depicted in figure 6(b).

3 A Pragmatic Guide to Fitting It All Together

In previous sections the core nine terms defining a Pattern Language for Memetic Algorithms were described. From an analysis of the literature and software available the following patterns define our language:
At the top of the conceptual hierarchy the Memetic Algorithm Pattern (MAP) appears. Its aim is to provide organisational principles for effective global-local search on hard problems. MAP can be implemented through a number of available (successful) templates some of which are based on Particle Swarm Optimisation, Ant Colony Optimisation, Evolutionary Algorithms, etc. Each of these gives rise to new terms in the language, namely, the EMATP (for the Evolutionary Memetic Algorithm Algorithmic Template), the ACOMATP (for the Ant Colony Memetic Algorithm Template), the SAMATP (for the Simulated Annealing Memetic Algorithm Template), etc. Each of these nature-inspired paradigm templates have their own "idiosyncrasies". However, at the time of implementing them as Memetic Algorithms the following common features are captured in new design patterns, namely, the Refinement Strategy Pattern (RSP), with focus on heuristic and local search methods, the Exact and Approximate Hybridisation Strategy Pattern (EAHSP) that defines ways in which expensive exact (or approximate) algorithms are integrated with a Memetic Algorithms Template Pattern under any nature-inspired realisation. Two other terms in our Pattern Language are the Population Diversity Handling Strategy Pattern (PDHSP), dealing with ways to preserve – in the face of aggressive refinement strategies – global diversity and the Surrogate Objective Function Strategy Pattern (SOFSP) whose role is to define methods for dealing within a Memetic Algorithm with very expensive/noisy/undetermined objective functions. Finally, the Pattern Language presented also include terms for defining the "generation" of Memetic Algorithm one is trying to implement. A Memetic Algorithm of the first generation in which only one refinement strategy is used, is called a Simple Strategy Pattern (SSP) and that is the canonical MAs (e.g. see Figure 3). The Multimeme Strategy Pattern (MSP) and the Self-Generating Strategy Pattern (SGSP), which provide methods for utilising several refinement strategies simultaneously and for synthetising new refinement strategies on the fly respectively, are second and third generation MAs. Thus the resulting Pattern Language has at least the fol-

lowing terms: { MAP, EMATP, PSOMATP, ACOMATP, SAMATP, . . ., AISMATP, RSP, EASHP, CPRSP, PDHSP, SOFSP, SSP, MSP, SGSP }.

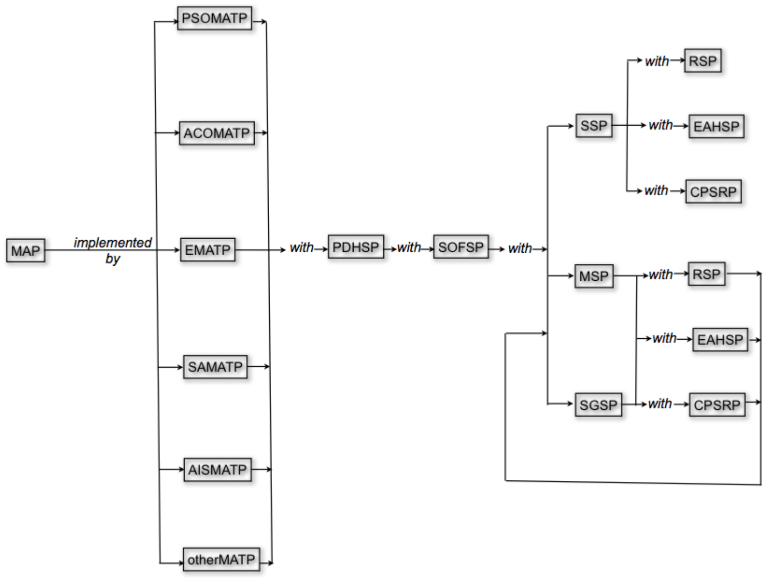


Fig. 7 Integrative view of the design patterns for the Memetic Algorithms Patterns Language. A path through the graph represents a series of design decision that an algorithms engineer would need to take while implementing a MA. Each design pattern provides solutions to specific problems the pattern addresses. By indexing through the patterns' acronyms into the main text in this chapter the reader can have access to examples from the literature where the issues have been solved to satisfaction.

These patterns are functionally related as depicted in Figure 7 that represents the series of design decisions involved in the implementation of a Memetic Algorithms. One could choose to implement a MA by following, e.g., an Ant Colony optimisation template or – more often – an Evolutionary Algorithm template or any of the other template patterns. Each one of these nature-inspired template patterns will have their own algorithmic peculiarities, e.g. crossovers and mutations for EAs, pheromones updating rules for ACOs, etc., but all of them when taken as a Memetic Algorithm will need to address the issues of population diversity, refinement strategies, exact algorithms, surrogate objective functions, single meme versus multimeme versus self-generation, etc. In order to instantiate code for any of these design patterns one can simply look at the description of the design pattern provided in this chapter and refer to any of the literature references given within the pattern description. Thus, Figure 7 provides a reference handbook for MAs engineering.

4 Brief Notes on Theoretical Aspects

Memetic Algorithms notorious successes in practical applications notwithstanding, no systematic effort has been made to try to understand them from a theoretical viewpoint. Memetic Algorithms do not adhere to a pure EAs theory, not even those applied to continuous problems for which convergence analysis are somewhat easier. Moreover, MAs are also much more complex than other heuristic methods such as greedy-like algorithms for which precise theoretical knowledge is possible. Thus while the body of empirical knowledge is steadily growing their theoretical underpinnings remain poorly understood. A collection of disjoint, but only partial, analyses were published in, e.g., [82, 132] for MAs applied to combinatorial optimisation and [50, 51] for continuous optimisation. Two emerging methodologies to assess heuristics performance that might in the future lead to important insights for MAs are PLS-theory [63, 143] and Domination analysis [45, 47]. Recent results based on both of these theoretical approaches to heuristic analysis are compiled in [99]. For an organisational perspective of the field the reader is also referred to the papers [82] and [111] in which several best practices are described and research lines proposed.

5 The Future: Towards Nature-Inspired Living Software Systems

Memetic Algorithms have gone through three major transitions in their evolution. The first transition was coincidental with their introduction and the bulk of the research and applications produced was based on the use of, mainly, an evolutionary algorithm with one specialised local searcher. The second transition brought the concept of multimeme algorithms that expanded the repertoire of both the key skeleton, i.e. EAs were not longer the only global search template used but rather any global (population) based search could be employed, and gave way to the constraint on having a single local searcher. The third major transition is taking place right now and involves the incorporation of explicit learning mechanisms that can, whilst optimising, "distill" new refinement operators. These new operators, in turn, can be reused in a later application of the MA, thus amortising the cost invested in discovering them. This third transition is also producing more frequent and more confident use of exact methods in tandem or hybridised with MAs. This third major transition is likely to produce very exciting results over the next few years and indeed much research remains to be done in the theory and practice of Memetic Algorithms.

The question that we would like to briefly address is "what's next?", "where do Memetic Algorithms go from here?".

Memetic Algorithms in particular and optimisation algorithms in general are but one class of the millions of different software systems humans create to solve problems. The vast majority – if not all – of software systems development follows, to various degrees of formalism and details, a process of requirements collection, specification, design, implementation, testing, deployment and long term maintenance. There are, indeed, a variety of software engineering techniques that advocate for the concurrent realisation or elimination altogether of (some of) these steps as a way to achieve a more agile software development. The unrelenting scaling up of available computational, storage and communication power is bringing closer the day when software will cease to be created following such a process and will, instead, be "planted/seeded" within a production environment (e.g. a factory, university, bank, etc) and then "grown" from the bottom-up into a fully developed functional software system. Although within the remit of optimisation problems, Self-Generating techniques such as [75, 126, 127, 76, 77, 73, 18, 19, 20, 114, 5, 40, 133] are moving incrementally in this direction perhaps an even more radical scenario can be imagined. Consider for example a given production environment, e.g. a factory, where a series of optimisation problems must be solved routinely such as personnel rostering (Π_{pr}), job-shop schedules (Π_{jss}), path planning (Π_{pp}), inventory management (Π_{in}), etc. Each of these problems, during long periods of time (weeks, months, years, even decades) give rise to large collections of problem instances (I_{Π_j} with $j \in \{\Pi_{pr}, \Pi_{jss}, \Pi_{pp}, \Pi_{im}\}$. Current practice dictates that for each Π a tailor made solver is used that in practice works well for the associated $I(\Pi)$. A key observation worth making is that the distribution of instances for each of the problems arising from a given factory (or bank, university, supermarket, airport, etc) is not random but rather strongly dependent on physical, legal, commercial and societal constraints operating over the factory. That is, the instances derived from each one of these problems, for a given factory, will have strong common features and similarities. This, in turn, provides a tremendous opportunity for a more organic and autonomic problem solving. What we have in mind can best be described as a new discipline of "Pluripotential Problem Solvers(PPS)". The idea behind Pluripotential Problem Solvers is that a minimum set of self-generating features would be hard-coded into a given pluripotential solver cell (PSC), then this software cell would be immersed (in multiple copies) into the factory environment and bombarded with problem instances over a period of time. After a while some of the PSC would differentiate along problem instance lines (very much as a stem cell differentiates along cell type lines). Once differentiated, a mature PSC (or solver cell for simplicity) would only be sensitive to new instances of the same problem. Furthermore, as time progresses, the solver cell would be able to incorporate new problem specific and instance-distribution specific problem solving capabilities (e.g. perhaps using some refinement of the technologies described in the Self-Generating Strategy Pattern) thus further specialising. In the end, the set of initially planted pluripotential solver cells would have been developing along very specialised lines whithin a specific production environment. Needless to say, if one were to transplant a specialised solver cell operating on, lets say, Π_{jss} in factory 1 as a solver for Π_{jss} in factory 2, the performance of the solver will be limited because – unless the distributions of instances generated in both factories are very similar – the specialised solver cell would not be able to cope with the new ones as it would have lost its capacity to specialise. What one would rather do is seed again in factory 2 a new set of PSC and let them mature within their production environment. The scenario described above is predicated under two key assumptions. The first one is that one does not expect to have a cutting-edge solver “out of the tin” (for which arguably one would be willing to pay considerable amounts of money) but rather than one would pay less (or nothing) for a PSC and be willing to wait until the software grows into a very specialised, niche-specific, cutting edge solver. The second assumption is that computational power, communication bandwidth and storage capacity will keep increasing while the price per floating point operation and gigabytes transmitted/stored will go on decreasing. The advent of Pluripotential Problem Solvers will require a concerted, creative and unorthodox research on the amalgamation of optimisation algorithms, data mining and machine learning as well as more in depth studies of nature-inspired principles that – complementing evolutionary approaches – so far have not been tapped for optimisation, namely, the creative power of self-assembly and self-organisation[78].

6 Conclusions

In this chapter we have provided an unorthodox introduction to Memetic Algorithms. We have analysed Memetic Algorithms as they appear in the literature and used in practice, rather than conforming to a top down definition of what is a Memetic Algorithm and then, based on a definition, prescribe how to implement them. The approach taken here allows for a pragmatic view of the field and provides a recipe book for creating Memetic Algorithms by resorting to a shared Pattern Language. This emerging Pattern Language for Memetic Algorithms serves as a catalog of parts (or concerns) that an algorithms engineer might want to resort to for solving specific design issues arising from the need to interlink global search and local search for hard complex problems. Due to space and time constraints, we have not considered design patterns for Multi Objective[21, 89] problems, parallelisation strategies [1, 103], etc., neither how these important new components of the Pattern Language would interact with the patterns described here. It is hoped that the Pattern Language for MAs presented in this chapter will, in the future, be expanded and refined by researchers and practitioners alike.

Acknowledgements The author would like to acknowledge the many friends and colleagues with whom he has collaborated over the years. Their ideas, scientific rigor and enthusiasm for Memetic Algorithms has been a continuous source of inspiration and challenges. He would also like to thank Jonathan Blakes and James Smaldon for their valuable comments during the preparation of this paper. The author wishes to acknowledge funding from the EPSRC for projects EP/D061571/1 and EP/C523385/1. Finally, he thanks the editors of this book for the opportunity they gave the author to contribute his thoughts to it.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation **6**, 443–462 (2002)
2. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language - Towns, Buildings, Construction. Oxford University Press (1977)
3. Armour, P.: The conservation of uncertainty, exploring different units for measuring software. Communications of the ACM **50**, 25–28 (2007)
4. Bacardit, J., Krasnogor, N.: Performance and efficiency of memetic Pittsburgh learning classifier systems. Evolutionary Computation **17(3)** (2009)
5. Bader-El-Den, M., Poli, R.: Evolving heuristics with genetic programming. In: GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 601–602. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1389095.1389212
6. Bader-El-Din, M.B., Poli, R.: Generating SAT local-search heuristics using a gp hyper-heuristic framework. In: LNCS 4926. Proceedings of the 8th International Conference on Artifcial Evolution, pp. 37–49 (2007)
7. Berrut, J.P., Trefethen, L.: Barycentric lagrange interpolation. SIAM Review **46**(3), 501–517 (2004)
8. Bhattacharya, M.: Surrogate based ea for expensive optimization problems. In: Proceedings for the IEEE Congress on Evolutionary Computation (CEC), pp. 3847–3854 (2007)
9. Blackmore, S.: The Meme Machine. Oxford University Press (1999)
10. Branke, J.: Creating robust solutions by means of an evolutionary algorithm. In: Parallel Problem Solving from NaturePPSN V, p. 119128 (1998)
11. Branke, J.: Reducing the sampling variance when searching for robust solutions. In: L.S. et al. (ed.) Proc. Genetic and Evolutionary Computation Conference, p. 235242. kl (2001)
12. Branke, J.: Evolutionary Optimization in Dynamic Environments. kl (2002)
13. Bull, L.: On model-based evolutionary computation. Journal of Soft Computing - A Fusion of Foundations, Methodologies and Applications **3**, 76–82 (1999)
14. Bull, L., Holland, O., Blackmore, S.: On meme–gene coevolution. Artificial Life **6**, 227–235 (2000)
15. Burke, E., Bykov, Y., Newall, J., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. IIE Transactions **36**, 509–528 (2004)
16. Burke, E., Gustafson, S., Kendall, G., Krasnogor, N.: Advanced population diversity measures in genetic programming. In: J.M. Guervos, P. Adamidis, H. Beyer, J. Fernandez-Villacanas, H. Schwefel (eds.) 7th International Conference Parallel Problem Solving from Nature, *Springer Lecture Notes in Computer Science*, vol. 2439, pp. 341–350. PPSN, sv, Granada, Spain (2002)
17. Burke, E., Gustafson, S., Kendall, G., Krasnogor, N.: Is increased diversity beneficial in genetic programming: An analysis of the effects on fitness. In: IEEE Congress on Evolutionary Computation, pp. 1398–1405. CEC, IEEE (2003)
18. Burke, E., Hyde, M., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: T. Runarsson, H.G. Beyer, E. Burke, J. Merelo-Guervos, D. Whitley, X. Yao (eds.) Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), LNCS 4193, pp. 860–869. sv (2006)
19. Burke, E., Hyde, M., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 1559–1565. ACM (2007)
20. Burke, E., Hyde, M., Kendall, G., Woodward, J.: Scalability of evolved on line bin packing heuristics. In: Proceedings of the Congress on Evolutionary Computation (CEC 2007), pp. 2530–2537 (2007)
21. Burke, E., Landa-Silva, J.: The design of memetic algorithms for scheduling and timetabling problems. In: W. Hart, N. Krasnogor, J. Smith (eds.) Recent Advances in Memetic Algorithms, pp. 289–312. sv (2004)

22. Burke, E., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for timetabling problems. European Journal of Operational Research **176**, 177–192 (2007)
23. Burke, E., Newall, J., Weare, R.: A memetic algorithm for university exam timetabling. In: E. Burke, P. Ross (eds.) The Practice and Theory of Automated Timetabling, *Lecture Notes in Computer Science*, vol. 1153, pp. 241–250. Springer Verlag (1996)
24. Burke, E., Newall, J., Weare, R.: Initialization strategies and diversity in evolutionary timetabling. Evolutionary computation **6**, 81–103 (1998)
25. Caponio, A., Cascella, G., Neri, F., Salvatore, N., Sumner, M.: A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives. IEEE Transactions On Systems, Man and Cybernetics - Part B **37**, 28–41 (2007)
26. Carr, R., Hart, W., Krasnogor, N., Burke, E., Hirst, J., Smith, J.: Alignment of protein structures with a memetic evolutionary algorithm. In: W. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. Potter, A. Schultz, J. Miller, E. Burke, N. Jonoska (eds.) GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufman (2002)
27. Cavalli-Sforza, L., Feldman, M.: Cultural Transmission and Evolution: A Quantitative Approach. Princeton University Press (1981)
28. Cheng, R., Gen, M.: Parallel machine scheduling problems using memetic algorithms. Computers & Industrial Engineering **33**(3–4), 761–764 (1997)
29. Cloak, F.: Is a cultural ethology possible. Human Ecology **3**, 161–182 (1975)
30. Cooper, J.: Java Design Patterns: a tutorial. Addison-Wesley (2000)
31. Cordon, O., Herrera, F., Stutzle, T.: A review on the ant colony optimization metaheuristic: Basis, models and new trends. MATHWARE AND SOFT COMPUTING **9** (2002)
32. Cutello, V., Krasnogor, N., Nicosia, G., Pavone, M.: Immune algorithm versus differential evolution: a comparative case study using high dimensional function optimization. In: Int. Conf. on Adaptive and Natural Computing Algorithms, ICANNGA 2007. LNCS, pp. 93–101. Springer, Berlin, Heidelberg, New York (2007)
33. Dawkins, R.: The Selfish Gene. Oxford University Press, New York (1976)
34. Dawkins, R.: The Extended Phenotype. Oxford, Freeman (1982)
35. Dorigo, M., , Gambardela, L.: Ant colony system: A cooperative learning approach to the travelling salesman problem. IEEE Transactions on Evolutionary Computation **1**(1) (1997)
36. Dueck, G.: New optimisation heuristics. the Great Deluge algorithm and record-to-record travel. Journal of Computational Physics **104**, 86–92 (1993)
37. Duque, T., Goldberg, D., K.Sastry: Improving the efficiency of the extended compact genetic algorithm. In: GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 467–468. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1389095.1389181
38. Durham, W.: Coevolution: Genes, Culture and Human Diversity. Stanford University Press (1991)
39. Fleurent, C., Ferland, J.: Genetic and hybrid algorithms for graph coloring. Annals of Operations Research **63**, 437–461 (1997)
40. Fukunaga, A.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation **16**(1), 31–61 (2008). DOI 10.1162/evco.2008.16.1.31. URL http://www.mitpressjournals.org/doi/abs/10.1162/evco.2008.16.1.31. PMID: 18386995
41. Gabora, L.: Meme and variations: A computational model of cultural evolution. In: L.Nadel, D. Stein (eds.) 1993 Lectures in Complex Systems, pp. 471–494. Addison Wesley (1993)
42. Gallardo, J., Cotta, C., Fernandez, A.: On the hybridization of memetic algorithms with branch-and-bound techniques. Systems, Man, and Cybernetics, Part B, IEEE Transactions on **37**(1), 77–83 (2007). DOI 10.1109/TSMCB.2006.883266
43. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Desing Patterns, Elements of Reusable Object-Oriented Sofware. Addison-Wesley (1995)
44. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling **9**(1), 7–34 (2006)

45. Glover, F., Punnen, A.: The traveling salesman problem: New solvable cases and linkages with the development of approximation algorithms. Journal of the Operational Research Society **48**, 502–510 (1997)
46. Gutin, G., Karapetyan, D., Krasnogor, N.: Memetic algorithm for the generalized asymmetric traveling salesman problem. In: M. Pavone, G. Nicosia, D. Pelta, N. Krasnogor (eds.) Proceedings of the 2007 Workshop On Nature Inspired Cooperative Strategies for Optimisation. Lecture Notes in Computer Science (LNCS), vol. to appear. Springer (2007)
47. Gutin, G., Yeo, A.: Domination analysis of combinatorial optimization algorithms and problems. In: Graph Theory, Combinatorics and Algorithms, Operations Research/Computer Science Interfaces, vol 34, pp. 145–171. sv (2006)
48. Hansen, P., Mladenovic, N.: Variable neighborhood search for the $p-$median. Location Sciences **5**(4), 207–226 (1998)
49. Hansen, P., Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research (130), 449–467 (2001)
50. Hart, W.: Locally-adaptive and memetic evolutionary pattern search algorithms. Evolutionary Computation **11**, 29–52 (2003)
51. Hart, W.: Rethinking the design of real-coded evolutionary algorithms: Making discrete choices in continuous search domains. Journal of Soft Computing - A Fusion of Foundations, Methodologies and Applications **9**, 225–235 (2005)
52. Hart, W., Krasnogor, N., Smith, J.: Recent Advances in Memetic Algorithms, *Studies in Fuzziness and Soft Computing*, vol. 166, chap. Memetic Evolutionary Algorithms, pp. 3–27. Springer Berlin Heidelberg New York (2004)
53. Hart, W.E.: Adaptive Global Optimization with Local Search. Ph.D. Thesis, University of California, San Diego (1994)
54. He, L., Mort, N.: Hybrid genetic algorithms for telecomunications network back-up routeing. BT Technology Journal **18**(4) (2000)
55. Hinton, G., Nowlan, S.: How learning can guide evolution. Complex Systems **1**, 495–502 (1987)
56. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1976)
57. Hoshino, S.: On davies, swann and campey minimisation process. The Computer Journal **14**, 426 (1971)
58. Houck, C., Joines, J., Kay, M., Wilson, J.: Empirical investigation of the benefits of partial lamarckianism. Evolutionary Computation 5(1): 31-60. (1997)
59. Ishibuchi, H., Kaige, S.: Implementation of simple multiobjective memetic algorithms and its application to knapsack problems. Int. J. Hybrid Intell. Syst. **1**(1-2), 22–35 (2004)
60. Jakob, W.: Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers needs. In: T.R. et al. (ed.) Proceedings of the IX Parallel Problem Solving From Nature Conference (PPSN IX). Lecture Notes in Computer Science 4193, pp. 132–141. Springer, Berlin, Heidelberg, New York (2006)
61. Jaszkiewicz, A.: Genetic local search for multi-objective combinatorial optimization. European Journal of Operational Research **137** (2002)
62. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing-A Fusion of Foundations, Methodologies and Applications **9**, 3–12 (2005)
63. Johnson, D., Papadimitriou, C., Yannakakis, M.: How easy is local search. Journal of Computer And System Sciences **37**, 79–100 (1988)
64. Jongen, H., Meer, K., Triesch, E.: Optimization Theory. sv (2004)
65. Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: a survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996)
66. Kallel, L., Naudts, B., Reeves, C.: Properties of fitness functions and search landscapes. In: L. Kallel, B. Naudts, A. Rogers (eds.) Theoretical Aspects of Evolutionary Computing, pp. 175–206. Springer, Berlin, Heidelberg, New York (2001)
67. K.Dowsland, E.Soubeiga, E.K.Burke: A simulated annealing hyper-heuristic for determining shipper sizes. European Journal of Operational Research **179**, 759–774 (2007)

68. Kirkpatrick, S., Gelatt, C., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
69. Kononova, A., Hughes, K., Pourkashanian, M., Ingham, D.: Fitness diversity based adaptive memetic algorithm for solving inverse problems of chemical kinetics. In: IEEE Congress on Evolutionary Computation (CEC), pp. 2366–2373. IEEE (2007)
70. Krasnogor, N.: Coevolution of genes and memes in memetic algorithms. In: A. Wu (ed.) Proceedings of the 1999 Genetic and Evolutionary Computation Conference Graduate Students Workshop Program (1999). URL http://www.cs.nott.ac.uk/ nxk/PAPERS/memetic.pdf. (poster)
71. Krasnogor, N.: Studies on the Theory and Design Space of Memetic Algorithms. Ph.D. Thesis, University of the West of England, Bristol, United Kingdom (2002). URL http://www.cs.nott.ac.uk/ nxk/PAPERS/thesis.pdf
72. Krasnogor, N.: Recent Advances in Memetic Algorithms, *Studies in Fuzziness and Soft Computing*, vol. 166, chap. Towards robust memetic algorithms, pp. 185–207. Springer Berlin Heidelberg New York (2004)
73. Krasnogor, N.: Self-generating metaheuristics in bioinformatics: the protein structure comparison case. Genetic Programming and Evolvable Machines **5**(2), 181–201 (2004)
74. Krasnogor, N., Blackburne, B., Hirst, J., Burke, E.: Multimeme algorithms for protein structure prediction. In: J.M. Guervos, P. Adamidis, H. Beyer, J. Fernandez-Villacanas, H. Schwefel (eds.) 7th International Conference Parallel Problem Solving from Nature, *Springer Lecture Notes in Computer Science*, vol. 2439, pp. 769–778. PPSN, Springer Berlin / Heidelberg, Granada, Spain (2002)
75. Krasnogor, N., Gustafson, S.: Toward truly "memetic" memetic algorithms: discussion and proof of concepts. In: D.Corne, G.Fogel, W.Hart, J.Knowles, N.Krasnogor, R.Roy, J.E.Smith, A.Tiwari (eds.) Advances in Nature-Inspired Computation: The PPSN VII Workshops. PEDAL (Parallel, Emergent and Distributed Architectures Lab). University of Reading. ISBN 0-9543481-0-9 (2002)
76. Krasnogor, N., Gustafson, S.: The local searcher as a supplier of building blocks in self-generating memetic algorithms. In: J.S. W.E. Hart, N. Krasnogor (eds.) Fourth International Workshop on Memetic Algorithms (WOMA4). In GECCO 2003 workshop proceedings (2003)
77. Krasnogor, N., Gustafson, S.: A study on the use of "self-generation" in memetic algorithms. Natural Computing **3**(1), 53 – 76 (2004)
78. Krasnogor, N., Gustafson, S., Pelta, D., Verdegay, J. (eds.): Systems Self-Assembly: Multidisciplinary Snapshots, *Studies in Multidisciplinarity*, vol. 5. Elsevier (2008)
79. Krasnogor, N., Pelta, D.: Fuzzy memes in multimeme algorithms: a fuzzy-evolutionary hybrid. In: J. Verdegay (ed.) Fuzzy Sets based Heuristics for Optimization. Springer (2002)
80. Krasnogor, N., Smith, J.: A memetic algorithm with self-adaptive local search: TSP as a case study. In: D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.G. Beyer (eds.) GECCO 2000: Proceedings of the 2000 Genetic and Evolutionary Computation Conference. Morgan Kaufmann (2000)
81. Krasnogor, N., Smith, J.: Emergence of profitable search strategies based on a simple inheritance mechanism. In: L. Spector, E. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshj, M. Garzon, E. Burke (eds.) GECCO 2001: Proceedings of the 2001 Genetic and Evolutionary Computation Conference. Morgan Kaufmann (2001)
82. Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: Model, taxonomy and design issues. IEEE Transactions on Evolutionary Algorithms **9**(5), 474–488 (2005)
83. Krasnogor, N., Smith, J.: Memetic algorithms: The polynomial local search complexity theory perspective. Journal of Mathematical Modelling and Algorithms **7**, 3–24 (2008)
84. Kretwski, M.: A memetic algorithm for global induction of decision trees. In: Proceedings of SOFSEM: Theory and Practice of Computer Science. Lecture Notes In Computer Science, pp. 531–540. sv (2008)
85. Kuhn, T.: The Structure of Scientific Revolution. University of Chicago Press (1962)
86. Landa-Silva, D., Le, K.N.: A simple evolutionary algorithm with self-adaptation for multi-objective optimisation, pp. 133–155. Springer (2008)

87. Landa Silva, J., E.K, B.: Using diversity to guide the search in multi-objective optimization, pp. 727–751. World Scientific (2004)
88. Lee, Z., Lee, C.: A hybrid search algorithm with heuristics for resource allocation problem. Information Sciences **173**, 155–167 (2005)
89. Li, H., Landa-Silva, D.: Evolutionary multi-objective simulated annealing with adaptive and competitive search direction. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008), pp. 3310–3317. IEEE Press (2008)
90. Liu, B.F., Chen, H.M., Chen, J.H., Hwang, S.F., Ho, S.Y.: Meswarm: memetic particle swarm optimization. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 267–268. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1068009.1068049
91. Liu, D., Tan, K.C., Goh, C.K., Ho, W.K.: A multiobjective memetic algorithm based on particle swarm optimization. Systems, Man, and Cybernetics, Part B, IEEE Transactions on **37**(1), 42–50 (2007). DOI 10.1109/TSMCB.2006.883270
92. Llora, X., Sastry, K., Yu, T., Goldberg, D.: Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: Proceedings of the 9th annual conference on Genetic and evolutionary Computation, pp. 1798–1805. ACM (2007)
93. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation **12**(3), 273–302 (2004)
94. Mayley, G.: Landscapes, learning costs and genetic assimilation. Evolutionary Computation **4**(3), 213–234 (1996)
95. McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous system. Bulletin of Mathematical Biophysics **5**, 115–133 (1943)
96. Merz, P.: The compact memetic algorithm. In: Proceedings of the IV International Workshop on Memetic Algorithms (WOMA IV). GECCO 2003. http://w210.ub.uni-tuebingen.de/portal/woma4/ (2003)
97. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In: New Ideas in Optimization, pp. 245–260. McGraw-Hill (1999)
98. Mezmaz, M., Melab, N., Talbi, E.G.: Combining metaheuristics and exact methods for solving exactly multi-objective problems on the grid. Journal of Mathematical Modelling and Algorithms **6**, 393–409 (2007)
99. Michiels, W., Aarts, E., Korst, J.: Theoretical aspects of local search. Monographs in Theoretical Computer Science. sv (2007)
100. Molina, D., Lozano, M., Garcia-Martines, C., Herrera, F.: Memetic algorithm for intense local search methods using local search chains. In: Hybrid Metaheuristics: 5th International Workshop. Lecture Notes in Computer Science, pp. 58–71. Springer, Berlin, Heidelberg, New York (2008)
101. Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., Olson, A.J.: Automated docking using a lamarkian genetic algorithm and an empirical binding free energy function. J Comp Chem **14**, 1639–1662 (1998)
102. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA (1989)
103. Nebro, A., Alba, E., Luna, F.: Multi-objective optimization using grid computing. Soft Computing **11**, 531–540 (2007)
104. Nelder, J., Mead, R.: A simplex method for function minimization. The Computer Journal **7**(4), 308–313 (1965). DOI 10.1093/comjnl/7.4.308
105. Neri, F., Jari, T., Cascella, G., Ong, Y.: An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Trans. Comput. Biol. Bioinformatics **4**(2), 264–278 (2007)
106. Neri, F., Tirronen, V., Karkkainen, T., Rossi, T.: Fitness diversity based adaptation in multimeme algorithms: A comparative study. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2374–2381. IEEE (2007)
107. Nguyen, Q.H., Ong, Y.S., Lim, M.H., Krasnogor, N.: A comprehensive study on the design issues of memetic algorithm. In: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, pp. 2390–2397. IEEE (2007)

108. Niesse, J., Mayne, H.: Global geometry optimization of atomic clusters using a modified genetic algorithm in space-fixed coordinates. Journal of Chemical Physics **105**(11), 4700–4706 (Sep. 15, 1996)
109. O'Neill, M., C.Ryan: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Genetic Programming (vol. 4). sv (2003)
110. Ong, Y., Keane, A.: Meta-lamarckian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation **8**, 99–110 (2004)
111. Ong, Y., Lim, M., Zhu, N., Wong, K.W.: Classification of adaptive memetic algorithms: A comparative study. IEEE Transactions On Systems, Man and Cybernetics - Part B **36**, 141–152 (2006)
112. Ong, Y., Lum, K., Nair, P.: Hybrid evolutionary algorithm with hermite radial basis function interpolants for computationally expensive adjoint solvers. Computational Optimization and Applications **39**, 97–119 (2008)
113. Paenke, I., Jin, J.: Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. IEEE Transactions on Evolutionary Computation **10**, 405–420 (2006)
114. Pappa, G., Freitas, A.: Discovering new rule induction algorithms with grammar-based genetic programming. In: O. Maimon, L. Rokach (eds.) Soft Computing for Knowledge Discovery and Data Mining, pp. 133–152. sv (2007)
115. Petalas, Y., Parsopoulos, K., Vrahatis, M.: Memetic particle swarm optimisation. Annals of Operations Research **156**, 99–127 (2007)
116. Pirkwieser, S., Raidl, G.R., Puchinger, J.: A lagrangian decomposition/evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In: C. Cotta, J. van Hemert (eds.) Recent Advances in Evolutionary Computation for Combinatorial Optimization, pp. 69–85. sp (2008)
117. P.Siepmann, Martin, C., Vancea, I., Moriarty, P., Krasnogor, N.: A genetic algorithm approach to probing the evolution of self-organised nanostructured systems. Nano Letters **7(7)**, 1985–1990 (2007)
118. Quang, Q., Ong, Y., Lim, M., Krasnogor, N.: Adaptive cellular memetic algorithm. Evolutionary Computation **17**, to appear (2009)
119. Raidl, G.R., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: C.B. et al. (ed.) Hybrid Metaheuristics - An Emergent Approach for Combinatorial Optimization, pp. 31–62. Springer, Berlin, Heidelberg, New York (2008)
120. Reeves, C.: Hybrid genetic algorithms for bin-packing and related problems. Annals of Operations Research **63**, 371–396 (1996)
121. Richerson, P., Boyd, R.: A dual inheritance model of the human evolutionary process: I. basic postulates and a simple model. Journal of Social and Biological Structures **I**, 127–154 (1978)
122. Romero-Campero, F., H.Cao, Camara, M., Krasnogor, N.: Structure and parameter estimation for cell systems biology models. In: M.K. et.al (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2008), pp. 331–338. ACM Publisher (2008)
123. Sacks, J., Welch, W., Mitchell, T., Wynn, H.: Design and analysis of computer experiments. Statistical Science **4** (1989)
124. Schwefel, H.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., New York, NY, USA (1993)
125. Smith, J.: Modelling GAs with self adaptive mutation rates. In: GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufman (2001)
126. Smith, J.: Co-evolution of memetic algorithms : Initial results. In: Merelo, Adamitis, Beyer, Fernandez-Villacans, Schwefel (eds.) Parallel problem solving from Nature - PPSN VII, LNCS 2439, pp. 537–548. sv (2002)
127. Smith, J.: Co-evolution of memetic algorithms for protein structure prediction. In: K. Hart, Smith (eds.) Proceedings of the Third International Workshop on Memetic Algorithms (2002)

128. Smith, J.: Co-evolving memetic algorithms: A learning approach to robust scalable optimisation. In: Proceedings of the 2003 Congress on Evolutionary Computation, pp. 498–505 (2003)
129. Smith, J.E.: Credit assignment in adaptive memetic algorithms. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1412–1419. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1276958.1277219
130. Smith, R., Smuda, E.: Adaptively resizing populations: Algorithms, analysis and first results. Complex Systems **1**(9), 47–72 (1995)
131. Sorensen, K., Sevaux, M.: MA:PM: memetic algorithms with population management. Computers and Operations Research **33**, 1214–1225 (2006)
132. Sudholt, D.: Memetic algorithms with variable-depth search to overcome local optima. In: Proceedings of the 2007 Conference on Genetic and Evolutionary Computation (GECCO), pp. 787–794. ACM Press (2007)
133. Tabacman, M., Bacardit, J., Loiseau, I., Krasnogor, N.: Learning classifier systems in optimisation problems: a case study on fractal travelling salesman problems. In: Proceedings of the International Workshop on Learning Classifier Systems, *Lecture Notes in Computer Science*, vol. (to appear). Springer (2008). URL http://www.cs.nott.ac.uk/ nxk/PAPERS/maxi.pdf
134. Tang, M., Yao, X.: A memetic algorithm for VLSI floorplanning. Systems, Man, and Cybernetics, Part B, IEEE Transactions on **37**(1), 62–69 (2007). DOI 10.1109/TSMCB.2006.883268
135. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers and Industrial Engineering **54**(3), 453–473 (2008)
136. Turney, P.: How to shift bias: lessons from the Baldwin effect. Evolutionary Computation **4**(3), 271–295 (1996)
137. Vavak, F., Fogarty, T.: Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In: Proceedings of the 1996 IEEE Conference on Evolutionary Computation, pp. 192–195 (1996)
138. Wang, H., Wang, D., Yang, S.: A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. Soft Computing **to appear** (2008)
139. Whitley, L., Gordon, S., Mathias, K.: Lamarkian evolution, the Baldwin effect, and function optimisation. In: Y. Davidor, H.P. Schwefel, R. Männer (eds.) PPSN, *Lecture Notes in Computer Science*, vol. 866, pp. 6–15. Springer (1994)
140. Whitley, L., Gruau, F.: Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evolutionary Computation **1**, 213–233 (1993)
141. Wolpert, D., Macready, W.: No Free Lunch theorems for optimisation. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
142. Yanga, J., Suna, L., Leeb, H., Qiand, Y., Liang, Y.: Clonal selection based memetic algorithm for job shop scheduling problems. Journal of Bionic Engineering **5**, 111–119 (2008)
143. Yannakakis, M.: Computational complexity. In: E. Aarts, J. Lenstra (eds.) Local Search in Combinatorial Optimization, pp. 19–55. John Wiley & Sons Ltd. (1997)
144. Zhou, Z.: Hierarchical surrogate-assisted evolutionary optimization framework. In: In Evolutionary Computation, 2004. CEC2004. Congress on, pp. 1586–1593 (2004)
145. Z.Zhou, Ong, Y., Lim, M., Lee, B.: Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Computing-A Fusion of Foundations, Methodologies and Applications **11**, 957–971 (2007)