Memetic Comp. (2009) 1:85–100
DOI 10.1007/s12293-009-0011-1

REGULAR RESEARCH PAPER

A proposition on memes and meta-memes in computing for higher-order learning

Ryan Meuth · Meng-Hiot Lim · Yew-Soon Ong · Donald C. Wunsch II

Received: 26 September 2008 / Accepted: 7 April 2009 / Published online: 29 April 2009

Abstract In computational intelligence, the term 'memetic algorithm' has come to be associated with the algorithmic pairing of a global search method with a local search method. In a sociological context, a 'meme' has been loosely defined as a unit of cultural information, the social analog of genes for individuals. Both of these definitions are inadequate, as 'memetic algorithm' is too specific, and ultimately a misnomer, as much as a 'meme' is defined too generally to be of scientific use. In this paper, we extend the notion of memes from a computational viewpoint and explore the purpose, definitions, design guidelines and architecture for effective memetic computing. Utilizing two conceptual case studies, we illustrate the power of high-order meme-based learning. With applications ranging from cognitive science to machine learning, memetic computing has the potential to provide much-needed stimulation to the field of computational intelligence by providing a framework for higher order learning.

Keywords Machine learning · Memetic computing · Meta-learning · Computational intelligence architectures

R. Meuth (✉) · D. C. Wunsch II
Applied Computational Intelligence Laboratory,
Department of Electrical and Computer Engineering,
Missouri University of Science and Technology, Rolla, MO, USA
e-mail: rmeuth@mst.edu

M.-H. Lim
School of Electrical and Electronic Engineering,
Nanyang Technological University, Singapore 639798, Singapore

Y.-S. Ong
School of Computer Engineering, Nanyang Technological University,
Singapore 639798, Singapore

1 Introduction

Over the past several years many hundreds of papers have been published on the modification and application of only a handful of core computational intelligence techniques—namely dynamic programming, evolutionary algorithms, neural networks, fuzzy logic, and data clustering methods. Algorithmically, there have been refinements and crossovers in these categories, such as heuristic dynamic programming, particle swarm optimization, evolutionary-trained fuzzy neural networks, and hybrid genetic algorithms, resulting in significant but relatively modest quality and performance gains. Beyond these modifications the pace of new algorithm design has been stagnant for a period of time, while the complexity of machine learning and optimization problems has grown ever larger with the maturity of the internet, digital media, and the proliferation of data sources in all aspects of human life.

Meanwhile, advancement in hardware technology has brought about affordable and powerful computing platforms which are more easily accessible. However, it is clear that increase in computational capacity cannot even come close to addressing the challenges posed by the complexity of problems, many of which are typical of real-world scenarios [14]. More advanced and novel computational paradigms, particularly from the algorithms front have to be championed. The general perception on how algorithms have managed to keep pace with increasing problem complexity over the years is depicted in Fig. 1. Initially, algorithms by and large were able to keep up with the demands of increasing problem complexity. To a certain extent, the algorithms which typically belong to the category of conventional or exact enumerative procedures were able to surpass the complexity of problems that were typical of what people were trying to solve. Subsequently, as the complexity of problems pushes the capability limits of algorithms, it became evident that the complexity

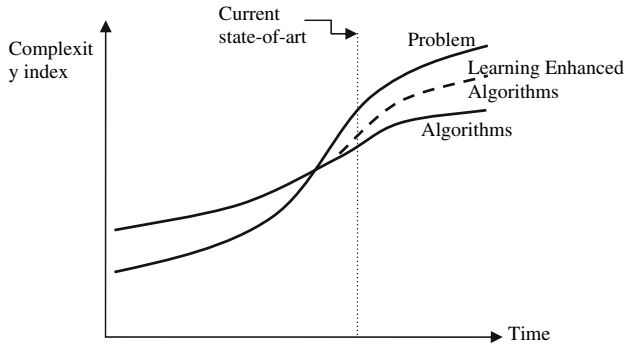

Fig. 1 An abstract comparison on state of optimization from the perspectives of problems and algorithms complexity

of problems being addressed began to overwhelm the algorithms available. We view the region corresponding to the convergence and divergence of the curves as being synonymous to the era of computational intelligence techniques. It can be envisaged that in time, the spread between complexity of problems and algorithms will widen if computational intelligence remains at status quo. There are clear signs that these issues are in the early stages of being addressed. In particular, the phase of research should be putting emphasis not just on learning per se, but rather on issues pertaining to higher order learning. This is a natural tendency in order to address the demands and challenges of problems that surface.

The era of computational intelligence to a certain extent managed to contain the gap between algorithms and problem. In time, it will become clear that the divergence between the two curves will continue, as shown in Fig. 1. A more promising outlook as shown by the broken line curve can be achieved and modern day optimization techniques can rise to this challenge by incorporating not just mechanisms for adaptation during the process of solving an instance of a difficult problem, but rather mechanisms of adaptation or more appropriately learning spanning across instances of problems encountered during the course of optimization. While a certain degree of similarity may be drawn when compared to case-based reasoning (CBR), such perceived 'experiential' trait similarity in the sense that both encompass mechanisms to draw on 'experience' from previously encountered problem instances is superficial. Unlike CBR methods which rely on the need for explicit examples and ranking procedures, optimization problems are usually not amenable to such explicit case by case assessment to yield information that is potentially useful to a search algorithm [23,70]. Rather, a more likely emphasis should be the building up of a body of knowledge, more specifically memes and meta-memes that collectively offer capability with a much broader problem-solving scope in order to deal with the class of problems being addressed.

In 1997, Wolpert and Macready formalized the 'No Free Lunch Theorem' stating simply:

> Any two [optimization] algorithms are equivalent when their performance is averaged across all possible problems.

Additionally, Wolpert and Macready made the observation that in order to reduce the average cost across a set of problems and optimizers, one must methodically utilize prior or acquired information about the matching of problems to procedures, given a priori knowledge gained from experience [71]. The realizations brought by the No Free Lunch Theorem changed the research focus of the field of computational intelligence from the design of individual algorithms to the design of architectures of algorithms and parameters optimization. It is in this spirit that the development of memetic algorithms has been motivated [13,21,31,32,34,36,41,53,54,73].

Taken alone, current methods tend to be overwhelmed by large datasets and suffer from the curse of dimensionality. A new class of higher order learning algorithms are needed that can autonomously discern patterns in data that exist on multiple temporal and spatial scales, and across multiple modes of input. These new algorithms can be architectures utilizing existing methods as elements, but to design these architectures effectively, some design principles should be explored.

Ultimately, the curse of complexity cannot be wholly avoided. As the size or dimension of the problems increases, a greater amount of computation becomes necessary to find high quality solutions. However, such computation need not be done on the fly, meaning at the exact time that a problem is presented. If a memory mechanism is provided that can store and retrieve previously used or generalized solutions, then computation can be shifted into the past, greatly reducing the amount of computation necessary to arrive at a high quality solution at the time of problem presentation.

One of the major drawbacks of evolutionary algorithms and computational intelligence methods in general is the solvers employed usually start from zero information, independent of how similar the problem instance is to other instances the method has been applied to in the past. In effect, the optimization methods typically do not incorporate any mechanisms to establish inter-instance memory. This property is useful for comparing different computational intelligence methods and in some cases, particularly when computation time is not an issue, the capacity to draw on memory of past instances solved is desirable as it allows the search to be more focused, thus leading to solutions that would not otherwise have been found efficiently. It is also worth noting that many real-world problem domains are composed of sub-problems that can be solved individually, and combined (often in a non-trivial way) to provide a solution for the larger problem [35,60].

In some problem instances, such as large instances of the even parity problem, it is nearly impossible to stochastically arrive at a complete solution without utilizing generalized

solutions for small instances of the problem [24]. It is simple to evolve a function that performs even parity on 2 bits using only the logical functions AND, OR and NOT as primitives, but extremely difficult to evolve a 10-bit even parity function without any a priori information as the space of all possible solutions is immensely larger, and even the best known solution is complex. By simply defining the general 2-bit XOR function (the even parity computation for 2 bits), the optimization method has a higher probability of combining instances of XOR to arrive at an n-bit even-parity function, greatly accelerating the optimization process.

In the game of chess, humans start at the top, and solve a successive sequence of smaller, tractable problems to arrive at a move. However, the learning process is bottom-up—a human player of chess first learns the legal moves of every piece, and then combines those general move capabilities into strategies, strategies into tactics and those tactics combine with the tactics of the opposing player to form a high-level view of the game as a whole. At each level optimization and generalization are performed to pass information up and down the play hierarchy. However, this natural progression is not reflected in the methods that we utilize to computationally approach problems of this scale. The typical approach is combinatorial optimization, where a sequence of low-level moves is statistically analyzed in order to arrive at a plan of play. As a whole, this is a computationally intractable problem, and it does not even come close to resembling the way humans play chess. Additionally, the skills learned in chess may translate across several domains as general problem solving skills. The ability to translate knowledge from one domain to another implies the necessity of meta-learning or learning about how or what to learn—in order to recognize similar problem features in disparate environments and scenarios.

The remaining of this paper is organized as follows. Section 2 gives a brief outline of the classes of brain inspired memetic computing. In Sect. 3 we discuss and compare between schema and memes, in particular their roles in learning. Section 1 gives an architectural framework for computing with memes and meta-memes, exposing some important issues in the design of systems with higher order learning capability. Two examples, the even parity in Sect. 5 and travelling salesman problem in Sect. 6 are studied to illustrate the concept of learning that spans across instances of problems. In Sect. 7, we conclude this paper.

2 Brain inspired memetic computing

While Darwinian evolution has been a source of inspiration for a class of algorithms for problem-solving, memetics has served as a motivation for problem-solving techniques with memetic algorithms being the most prominent and direct manifestation of the inspiration. In recent years, there has been a marked increase in research interests and activities in the field of Memetic Algorithms. The first generation of MA refers to hybrid algorithms, a marriage between a population-based global search (often in the form of an evolutionary algorithm) coupled with a cultural evolutionary stage. The first generation of MA though it encompasses characteristics of cultural evolution (in the form of local refinement) in the search cycle, may not qualify as a true evolving system according to Universal Darwinism, since all the core principles of inheritance/memetic transmission, variation and selection are missing. This suggests why the term MA stirs up criticisms and controversies among researchers when first introduced in [43]. The typical design issues [49] include (i) how often should individual learning be applied, (ii) on which solutions should individual learning be used, (iii) how long should individual learning be run, (iv) what maximum computational budget to allocate for individual learning, and (v) what individual learning method or meme should be used for a particular problem, sub-problem or individual.

Multi-meme [28], hyper-heuristic [22] and meta-Lamarckian MA [53,54] are referred to as second generation MA exhibiting the principles of memetic transmission and selection in their design [48]. In multi-meme MA, the memetic material is encoded as part of the genotype. Subsequently, the decoded meme of each respective individual is then used to perform a local refinement. The memetic material is then transmitted through a simple inheritance mechanism from parent to offspring. On the other hand, in hyper-heuristic and meta-Lamarckian MA, the pool of candidate memes considered will compete, based on their past merits in generating local improvements through a reward mechanism, deciding on which meme to be selected to proceed for future local refinements. A meme having higher rewards will have greater chances of being replicated or copied subsequently. For a review on second generation MA, i.e., MA considering multiple individual learning methods within an evolutionary system, the reader is referred to [53]. Co-evolution and self-generation MAs introduced in [34] and [62] are described in [48] as third generation MA where all three principles satisfying the definitions of a basic evolving system has been considered. In contrast to second generation MA which assumes the pool of memes to be used being known a priori, a rule-based representation of local search is co-adapted alongside candidate solutions within the evolutionary system, thus capturing regular repeated features or patterns in the problem space.

From the three classes of MA outlined, memes can be seen as mechanisms that capture the essence of knowledge in the form of procedures that affect the transition of solutions during a search. The level of participation or activation of memes is typically dictated by certain indicative performance

Table 1 Generational descriptions of memetic algorithms

Classes	Characteristics	Example systems
First Generation	Global search paired with local search	(i) A canonical MA [43,50] (ii) Adaptive global/local search [16] (iii) MA for combinatorial optimization [33] (iv) Handling computationally expensive problems [55] (v) Multiobjective permutation flowshop scheduling [19] (vi) Fitness landscape analysis of MA [38] (vii) Robust aerodynamic design [56] (viii) Evolutionary gradient search (Arnold and Salomon [6]) (ix) Large-scale quadratic assignment problem [63] (x) Evolutionary Lin–Kernighan for traveling salesman problem [40] (xi) Dynamic optimization problem [68] and many others
Second Generation	Global search with multiple local optimizers. Memetic information (Choice of optimizer) Passed to offspring (Lamarckian evolution)	(i) Nurse rostering problem [9] (ii) Hyper-heuristic MA [15,22] (iii) Structure prediction and structure comparison of proteins [29] (iv) Meta-Lamarckian MA [54] (v) Multimeme MA [31] (vi) Adaptive multi-meme MA [53] (vii) Multimeme algorithm for designing HIV multidrug therapies [10,45] (viii) Agent-based memetic algorithm [17,67] (ix) Diffusion memetic algorithm [48] and several others
Third Generation	Global search with multiple local optimizers. Memetic information (Choice of local optimizer) passed to offspring (Lamarckian Evolution). A mapping between evolutionary trajectory and choice of local optimizer is learned	(i) Co-evolution MA [62] (ii) Self-generation MA [30]
4th Generation	Mechanisms of recognition, Generalization, optimization, and memory are utilized	Unknown

metrics, the objective being to achieve a healthy balance between local and global search. Memes instead of being performance-driven should be extended to include capacity to evolve based on the snapshots of problem instances. In the process of solving a repertoire of problem instances, memes can culminate based on the recurrence of patterns or structures. From basic patterns or structures, more complex higher level structures can arise. In this regard, a brain inspired meta-learning memetic computational system, consisting of an optimizer, a memory, a selection mechanism, and a generalization mechanism that conceptualizes memes not just within the scope of a problem instance, but rather in a more generic contextual scope is appropriate. Such traits which are lacking in the third generation MA can serve as the basis of 4th generation class of MAs. The reader is referred to Table 1 for a summary of generational description of Memetic Algorithms. The summary although by no means exhaustive should serve as a useful guide on the classifications of the various traits of existing MA research.

The mammalian brain exhibits hierarchical self-similarity, where neurons, groups of neurons, regions of the brain, and even whole lobes of the brain are connected laterally and hierarchically. Biological neurons are particularly well suited to this architecture; a single neuron serves as both a selection and learning mechanism. A neuron only fires when it receives significant input from one or more sources, and thus serves as a correlation detector. Additionally, it learns by modifying the weights of its inputs based on local information from firing rate, as well as global information from the chemical environment. Thus neurons activate when they encounter patterns that have made them fire before, and are able to adapt in delayed-reward situations due to global signals.

In laterally connected architectures, neuron groups can provide the function of clustering, as active neurons suppress the activity of their neighbors to pass their information down the processing chain, providing both selection and routing of information. The effect of this selectivity is that biological neural architectures route a spreading front of activation to different down-stream networks based on the similarity of the features present in the pattern of activation to previously presented patterns. As the activation front passes each neuron, the synaptic weights are changed based on local information—the firing rate of the neuron, the chemical environment, and the features present in the signal that activated the neuron, slightly changing how an individual neuron will respond at the next presentation of patterns [8].

Connected in loops, neurons provide short-term memory, process control and create temporally-delayed clustering. Combining loops and lateral connections at several levels of neuron groups (groups of neurons, groups of groups, etc) the neural architecture is able to exhibit increasing levels of selection, memory, and control. This is exactly the architecture that we see in the human cortex—a single cortical column contains recursion and lateral inhibition, and these cortical columns are arranged in a similar way, progressing in a fractal learning architecture up to the level of lobes, where sections of the brain are physically separated [20]. This fractal architecture is similar to the Nth-order meta-learning architecture described later in Sect. 4.

The brain inspired meta-learning memetic computational system is thus regarded here as a 4th generation memetic computational system. The novelty of the proposed meta-learning memetic system is highlighted below.

(i) In contrast to the second generation memetic algorithms, there is no need to pre-define a pool of memes that will be used to refine the search. Instead memes are learned automatically—they are generalized information that passed between problem instances.
(ii) Since it satisfies all the three basic principles of an evolving system, it also qualifies as a third generation memetic computational system. Unlike simple rule-based representation of meme used in co-evolution and self-generation MAs, the brain inspired meta-learning memetic computational system models the human brain that encodes each meme as hierarchies of cortical neurons [20]. With a self-organizing cortical architecture, meaningful information from recurring real-world patterns can be captured automatically and expressed in hierarchical nested relationships. A human brain stimulated by the recurrence of patterns, builds bidirectional hierarchical structures upward. The structure starts from the sensory neurons, through levels of cortical nodes and back down towards muscle activating neurons.
(iii) There exists a memory component to store the system's generalized patterns or structures of previously encountered problems—these elements could be thought of as memes.
(iv) Selection mechanisms are provided to perform association between problem features and previously generalized patterns that are likely to yield high-quality results.
(v) Meta-learning about the characteristics of the problem is introduced to construct meta-memes which are stored in the selection mechanism, allowing higher-order learning to occur automatically.
(vi) Memes and meta-memes in computing are conceptualized for higher-order learning as opposed to the typical definition of local search method used in all the works in memetic algorithm.

3 Schema–meme relationship

A genetic algorithm learns by passing schema (the genetic information of individuals) from generation to generation. Through natural selection and reproduction, useful schemata proliferate and are refined through genetic operators. The central concept of learning is that of the schema—a unit of information that is developed through a learning process [18,57,59]. The typical 'memetic algorithm' uses an additional mechanism to modify schemata during an individual's 'lifetime', taken as the period of evaluation from the point of view of a genetic algorithm, and that refinement is able to be passed on to an individual's descendants. The concept of schemata being passable just as behaviors or thoughts are passed on is what we term as memes—a meme being a unit of cultural information [53,54,61,64]. Thus, memes can be thought of as an extension of schemata—schemata that are modified and passed on over a learning process. However, this distinction is a matter of scale. In a learning method, the current content of the representation could be called a schema, but when that information is passed between methods, it is more appropriately regarded as a meme.

This is analogous to the sociological definition of a meme [12]. In this form, a meme may contain certain food preparation practices, or how to build a home or which side of the road to drive on. Within the individuals of a generation, they are relatively fixed, but they are the result of a great deal of optimization, capturing the adaptations resulting from the history of a society. These cultural memes are passed from generation to generation of the population, being slightly refined at each step—new ingredients are added to the cooking methods, new building materials influence construction, traffic rules change, etc. The mechanism that allows this transformation is that of generalization [51,52,58].

To communicate an internal schema from one individual to another, it must be generalized into a common representation—that of language in the case of human society. The specifics of the schema are of no great importance, as they would mean very little to an individual other than the originator due to the inherent differences between individuals. For instance, a description of the precise movements necessary to create a salad, such as the technique used to slice tomatoes and wash lettuce, is less important than the ingredients and general process of preparing the salad. Thus the salad recipe is a meme, a generalized representation of the salad, but the recipe alone is insufficient to produce the salad. The salad recipe is expressed only when it is put through the process of preparation, of acquiring and preparing the individual ingredients, and combining them according to the salad meme.

A meme may be thought of as generalized schema. Schemata are refined for an instance; memes are generalized to the extent of being transmissible between problem instances. To resolve the potential confusion that may arise, we put forth a loose definition of the term 'Memetic Computation'—a paradigm of computational problem-solving that encompasses the construction of a comprehensive set of memes thus extending the capability of an optimizer to quickly derive a solution to a specific problem by refining existing general solutions, rather than needing to rediscover solutions in every situation.

4 A framework for higher order learning

A meta-learning system should be composed of four primary components—an optimizer, a memory, a selection mechanism, and a generalization mechanism, shown in Fig. 2. The selection mechanism takes the features of a given problem as input, and performs a mapping to solutions in the memory that have an expected high quality. The memory stores previous or generalized solutions encountered by the system, and passes selected solution(s) on to the optimizer. The optimizer performs specialization and modification of solutions to optimize a given specific problem instance, while the generalization mechanism compares the resultant solution with existing solutions in memory, and either adds a new solution or modifies an existing solution. In memetic computation terms, the optimizer generates or modifies memes into schema, and then the generalization mechanism converts the schema back into memes for storage in memory. The selection mechanism provides a mapping about memes, providing recognition from a problem specification to a likely useful general solution, effectively utilizing internally represented meta-memes.

With these components, the architecture should be capable of exploiting information gained in previous problem

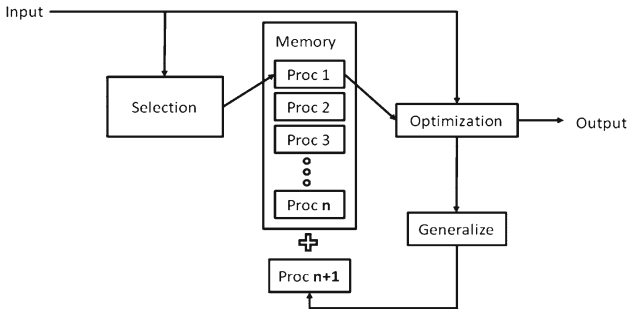

Fig. 2 Meta-learning architecture

sessions towards the solution of problems of increasing complexity. Integrating a cross-instance memory and a selection mechanism with an optimization method allows the recognition of a situation and the selection of previously utilized schema as likely high quality solution candidates. The optimization process then combines and refines these solution candidates to provide a good solution much faster than if the method had only random initial solutions. Once the solution is deployed, the selection method is trained to associate the situation (stimulus) with the solution (behavior) utilizing the fitness (reward) of the solution.

The process described above is itself a learning process, and thus could be augmented with increasingly higher level memory and selection methods, to allow complex, high-order solutions to be found. A sort of fractal meta-learning architecture of this type may be capable of virtually unlimited problem-solving capacity across a wide variety of problem domains.

The sequence of learning sessions matters greatly to the expression of complex behavior. By starting with simple problem instances and presenting successively more complex scenarios, the problem is decomposed, allowing solutions from sub-problems to be exploited, increasing the likelihood that higher level solutions will occur. Additionally, by training these simple solution components, a wider variety of high-level solutions can be trained more rapidly. For example, when training a dog, teaching him to 'sit' decreases the amount of training necessary for both 'stay' and 'beg' behaviors. This is analogous to the automatic construction of a 'Society of Mind' as described by [42].

When constructing optimization architectures, an issue of particular relevance is that of representation—how the schemata are stored. In population based algorithms schemata are stored as parameter strings, in neural networks, schemata are implicitly represented as interconnection weights, clustering methods store templates for categories, etc. How these schemata are expressed (and thereby their meaning) is dependent on the expression structure. In genetic algorithms a string is decoded into a trial problem solution, the weights

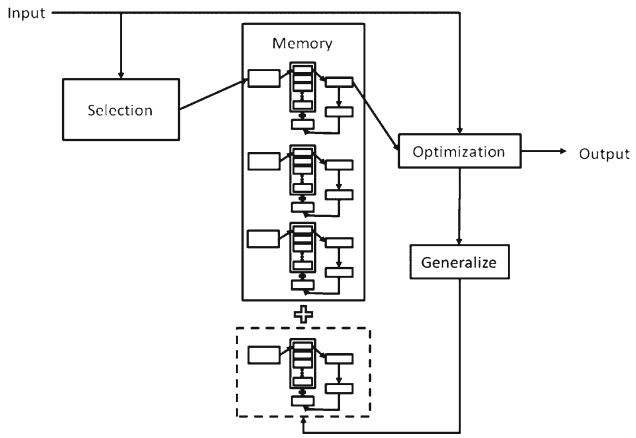

Fig. 3 Meta-meta learning

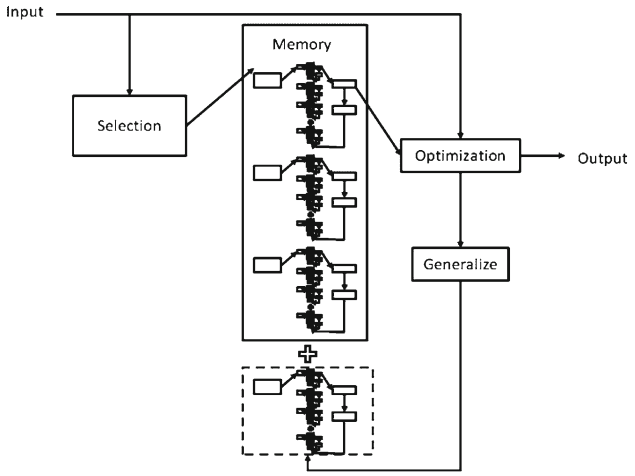

Fig. 4 *N*th-order meta learning

in neural networks are utilized through weighted summation and passing through a transfer function. This division of representation prevents the simple utilization of schema across solution methods. To get disparate methods to work together, great care must be taken to modify both methods to utilize the same schema, which has been the subject of a great deal of research [1,2,4,7,11,25,39,44,46,54].

First order learning methods consist of a single algorithm that modifies schema to optimize a system. Individually, all classical machine learning methods fall into this category. Meta-learning or second-order methods learn about the process of learning, and modify the learning method, which in turn modifies schema. A simple illustration of a meta-learning architecture is presented in Fig. 2. In this figure, schemata are represented as 'procedures', which are stored in memory. A problem is presented to the architecture, and a selection mechanism chooses likely valuable schema from memory, which are then modified to the particular problem instance. High-value schema are then generalized and saved back into memory, and the selection mechanism then learns an association between characteristics of the problem instance and schema that yielded positive results.

These second order methods should be able to be combined with other methods or layers to produce third-order methods and so on to order N, as illustrated in Figs. 3 and 4. To produce higher order methods, information gained in one problem instance should be utilized to provide a partial solution to another similar problem instance allowing the system as a whole to take advantage of previous learning episodes.

5 Even-parity example

To demonstrate the principles and advantages of meta-learning, we examine its application to the even and odd parity problems, standard benchmarks for genetic programming and automatic function definition methods [26,27]. We propose a hypothetical genetic programming system utilizing a set of Boolean operators to construct individuals implementing the even or odd parity functions (XOR and XNOR, respectively). We analyze two cases of the evolution of the three-input XOR function, both starting with populations implementing the two-input XOR function, with and without the abstraction that is inherent in a meta-learning system. A third case is presented illustrating the functionality of a simple selection mechanism on the odd-parity function.

5.1 Problem overview

Koza described the even parity problem below.

> The Boolean even-parity function of k Boolean arguments returns T (True) if an odd number of its arguments are T, and otherwise returns NIL (False). The concatenation of this returned bit to the original string making the total string even, hence even-parity.
> In applying genetic programming to the even-parity function of k arguments, the terminal set T consists of the k Boolean arguments $D0$, $D1$, $D2$, ... involved in the problem, so that
>
> $T = \{D0, D1, D2, \ldots\}$.
>
> The function set F for all the examples herein consists of the following computationally complete set of four two-argument primitive Boolean functions:
>
> $F = \{\text{AND, OR, NAND, NOR, NOT}\}$.
>
> The Boolean even-parity functions appear to be the most difficult Boolean functions to find via a blind random generative search of expressions using the above function set F and the terminal set T. For example,

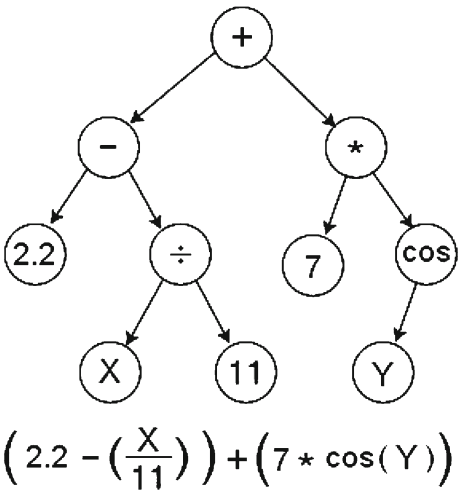

Fig. 5 Illustration of function representation as tree structure

even though there are only 256 different Boolean functions with three arguments and one output, the Boolean even-3-parity function is so difficult to find via a blind random generative search that we did not encounter it at all after randomly generating 10,000,000 expressions using this function set F and terminal set T. In addition, the even-parity function appears to be the most difficult to learn using genetic programming using this function set F and terminal set T [26,27].

The odd-parity function is similarly constructed, returning true if an even number of its arguments are true, and otherwise returning false.

In genetic programming (GP), the genome of an individual is represented as a tree structure, where operations are applied at branches, and the leaves are constants and problem parameters. An illustration of a functional represented as tree strurture is shown in Fig. 5 [24,26,27]. One advantage of GP is that the results can be easily human interpretable and formally verifiable, a quality that is not present in many other computational intelligence methods [58].

The even-2-parity function is simply the XOR function, which is itself a composition of the terminal set functions in one simple possible configuration:

a XOR b = (a OR b) AND (a NAND b)

Using a tree representation, the XOR function is shown in Fig. 6.

Constructing the even-3-parity function using only these primitives is more difficult, but follows a similar pattern, illustrated below and in Fig. 7:

XOR (a, b, c) = (((a OR b) AND (a NAND b)) OR c)
AND (((a OR b) AND (a NAND b)) NAND c)

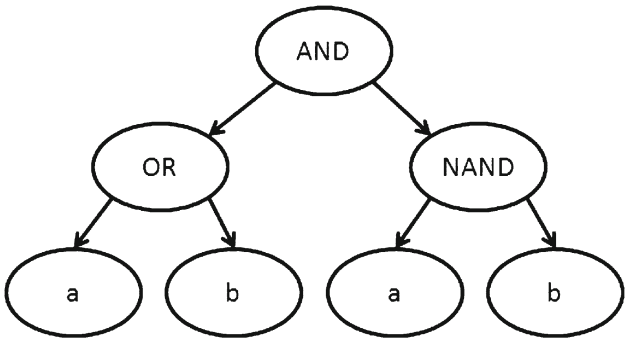

Fig. 6 XOR tree representation

Note that the three-input XOR structure relies on the recursive use of the two-input XOR function, replacing the 'a' nodes with XOR nodes, and re-assigning the top-level 'b' nodes to be the 'c' variable.

Note that if a 2-bit XOR function is defined explicitly as in Fig. 8, the even-3-parity function becomes greatly simplified, as written below and shown in Fig. 9.

XOR (a, b, c) = (a XOR b) XOR c

5.2 Case 1: Non-meta XOR3 evolution

Taking a genetic programming system as an example, in a non-meta learning system, evolution of the XOR3 function must proceed through at least two generations. To further expand on our illustration, we consider the best case scenario whereby all the individuals in the population incorporate the simplified XOR function, as shown in Fig. 10.

As there are four leaf nodes out of seven total nodes, the probability of selecting a leaf node for crossover (P_{L1}) is 4/7. Assuming a uniform population of individuals implementing XOR2 (translating to a 100% probability of choosing another XOR2 individual for crossover) the probability of selecting the root node of another individual to replace the selected leaf node is (P_{F1}) 1/7.

Then, the evolutionary process must select one of the two top-level 'b' nodes for mutation from the tree which has a total of 13 nodes, thus the probability of selecting one correct leaf for mutation (P_{M1}) is 2/13. Choosing from the eight possible node types (the combination of terminal set and functional set), the probability of selecting the correct 'c' variable (P_{V1}) is 1/8.

At this point the evolutionary reproduction steps are completed, and the individual shown in Fig. 11 is evaluated. This partial XOR3 function is not yet complete, but it correctly completes one test case more than the XOR2 function, which may give it an evolutionary advantage. Assuming that the individual survives to the next generation and is again selected as a parent with 100% probability, an additional

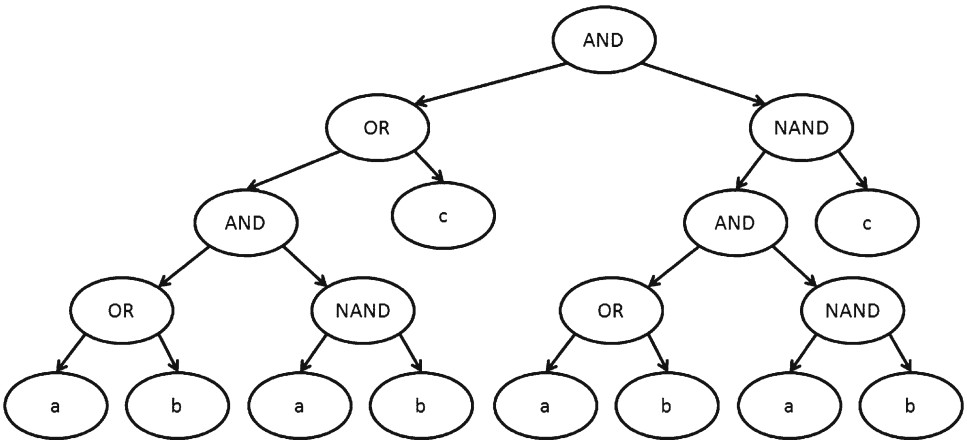

Fig. 7 Three-input XOR tree representation

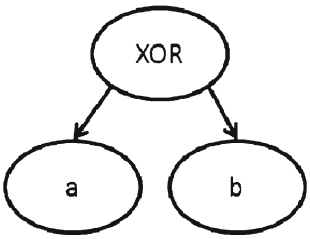

Fig. 8 Simplified two-input XOR

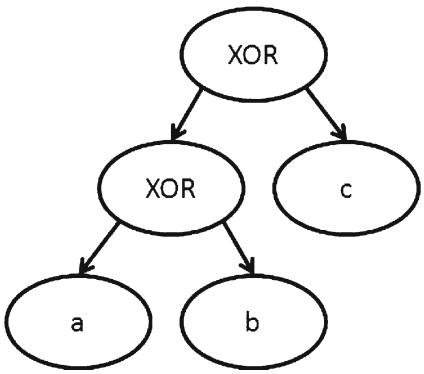

Fig. 9 Simplified three-input XOR

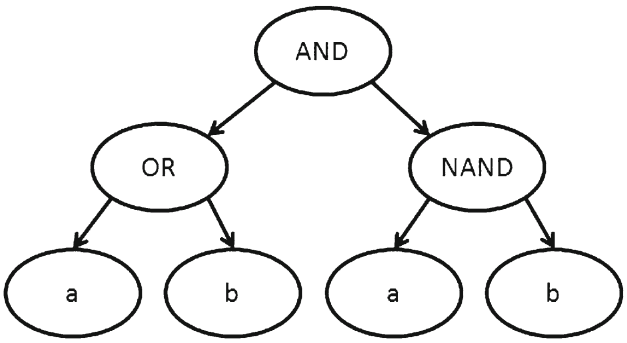

Fig. 10 Initial non-meta learning XOR2 individual

reproduction step must be completed to yield an XOR3 function.

Now the correct leaf node must be selected for crossover, but this time there is only one node, the 'a' node at a depth of three, from the 13 possible nodes, so the probability of selecting the correct leaf node for crossover (P_{L2}) is 1/13. Once again, assuming all other individuals in the population still implement the XOR2 function in Fig. 8, the probability of selecting the root of another XOR2 individual to replace the leaf (P_{F2}) is 1/7. At the completion of crossover, the total number of nodes in the tree becomes 18. At the mutation step, the remaining 'b' node at depth three must be selected, and the probability of selecting correct leaf for mutation (P_{M2}) is 1/18. Completing the XOR3, the probability of selecting the correct variable from the total set of node types (P_{V2}) is 1/8. The completed three-input XOR function is illustrated earlier in Fig. 9.

Ignoring changes in the population and evolutionary survivability, the probability of transitioning from XOR2 to XOR3 in two generations without meta-learning is calculated below.

$$\begin{aligned} P_{\text{xor3\_nonmeta}} &= P_{L1} * P_{F1} * P_{M1} * P_{V1} * P_{L2} * P_{F2} \\ &\quad * P_{M2} * P_{V2} \\ &= 1.19 \times 10^{-7} \end{aligned}$$

where,

P_{L1} the probability of a leaf node selection for crossover during the first generation,
P_{F1} the probability of functional root selection for crossover during the first generation,
P_{M1} the probability of proper leaf selection for mutation during the first generation,
P_{V1} the probability of proper variable selection for mutation during the first generation,
P_{L2} the probability of a leaf node selection for crossover during the second generation,

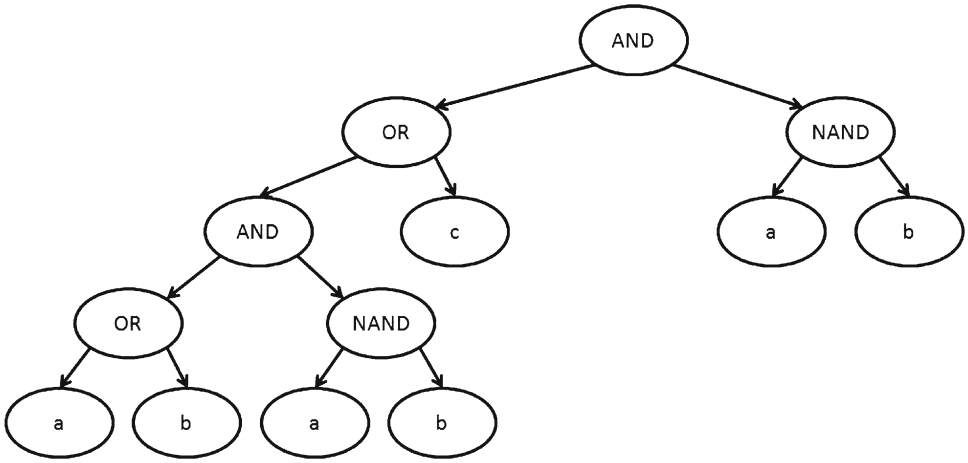

Fig. 11 Intermediate step in development of 3-bit XOR function after a single generation

P_{F2} the probability of functional root selection for crossover during the second generation,
P_{M2} the probability of proper leaf selection for mutation during the second generation,
P_{V2} the probability of proper variable selection for mutation during the second generation.

Note that this ignores the significant influence of relative fitness, generational selection, parent selection, probability of application of crossover/mutation operators and population influence and may be interpreted as a kind of upper-bound on the probability that a two-input XOR individual will develop into a three-input XOR without the abstraction capability of meta-learning.

5.3 Case 2: Meta-learning XOR3 evolution

In this case we assume a meta-learning system that has already learned a two-input XOR function, performed generalization and added this to the function set (F = AND, OR, NAND, NOR, NOT, XOR2). The probability that the system will transition from XOR2 to XOR3 is calculated using only the mutation step.

With a population uniformly initialized with the two-input XOR and an individual selected from this population, illustrated in Fig. 8, the probability of selecting a leaf node for mutation (P_L) is 2/3 as the simplified XOR tree has only three nodes, and two of them are terminals. Having selected a terminal, the probability of selecting the XOR2 function from the node set of six functions and three terminals to replace the leaf node (P_F) is 1/9. Assuming a recursive mutation process, two new leaf nodes must be selected, and they must contain variables not yet used by the tree to produce a three-input XOR. The probability of selecting the correct terminal node is 1/9, and this process must be repeated twice, so the probability of selecting two correct terminal nodes (P_V) is $(1/9)^2$ or 1/81. Using only one generation the three-input XOR can be developed in a meta-learning system.

$$\text{Probability of XOR3 from XOR2} : P_{\text{xor3\_meta}} = P_L * P_F * P_V = 0.000914$$

where,

P_L the probability of a leaf node selection for mutation,
P_F the probability of XOR2 function selection for mutation,
P_V the probability of proper leaf selection for mutation.

Note that using meta-learning, the three-input XOR can also occur with a crossover and a mutation, where the non-meta learning system must utilize two full generations. Also note that though the size of the functional set has increased, the number of changes necessary to place an upper-bound on the probability of a three-input XOR occurring has been substantially decreased, allowing the evolutionary process to focus on high-level changes.

Thus in a large population, the XOR3 function may occur in a single generation with a meta-learning system, where a non-meta learning system must take at least two generation and probably many thousands of evaluations to evolve an XOR3.

5.4 Case 3: Selection and odd-parity evolution

To demonstrate the advantages of the complete meta-learning procedure, we first present the 2-bit even-parity problem to a theoretical meta-learning system, then the 2-bit odd-parity problem, and finally the 3-bit even-parity problem. The selection mechanism shall have 2 inputs—the first is activated only when the system is operating on the even-parity problem, the second is activated only when operating on the odd-parity problem. Initially, the memory is empty, so the optimizer is initialized with random solutions.

Presented with the even-2-parity problem, the optimizer outputs a resulting solution that performs the XOR function—'D0 XOR D1', where D0 and D1 are the Boolean arguments of the input. This function is passed to the generalization mechanism, which removes the absolute references to the Boolean arguments, replacing them with dummy variables 'A' and 'B', resulting in the function 'A XOR B'. This generalized XOR function is then added to the memory, making the function available as a primitive. The functional set becomes:

$F = \{\text{AND, OR, NAND, NOR, NOT, XOR}\}$.

The selection mechanism is updated to learn an association between the active 'even-parity' input and the new memory element. At this point the procedure and difference in optimization would be no different than if the optimizer were operating without the rest of the meta-learning architecture.

Next, the odd-2-parity problem is presented, the 'odd-parity' input is activated on the selector mechanism, and having no other elements to select, the sole item in memory (the generalized 'A XOR B' function) is selected to initialize the state of the optimizer. The optimizer replaces the dummy variables with references to the Boolean arguments and begins optimization. As only a small modification is necessary, the addition of the NOT primitive function at a high-level to create an XNOR function, the optimizer has a high probability of quickly finding a perfect solution to the odd-2-parity problem. This differs from a randomly initialized optimizer as there would be a lower probability of finding a good solution due to the need to explore more modifications. Once the meta-learning optimizer finds the solution, the generalization, memory insert, and selection training steps are repeated for the XNOR function:

$F = \{\text{AND, OR, NAND, NOR, NOT, XOR, XNOR}\}$.

Finally, the even-3-parity problem is presented to the meta-learning architecture. The selection 'even-parity' input is activated, and the associated XOR memory element is used to initialize the optimizer state. The optimizer replaces the XOR dummy variables with argument references, and begins the optimization process. The optimizer need only make the relatively small change of cascading the XOR function to produce a 3-input XOR function, where a raw optimization function without a memory or selection method would need to evaluate and modify many combinations of the original five functional primitives to arrive at a good solution. Thus the meta-learning architecture should be able to arrive at high-value solutions rapidly by exploiting previously generated solution to construct high-level solutions.

In this example the memory component stores generalized solutions to previously encountered problems—these elements could be thought of as memes, as they are solutions that are passed between problem instances. The selection mechanism performs association between problem features and solutions that are likely to yield high-value results. By not only providing the input data to the problem, but additional meta-data about the characteristics of the problem, the meta-learning architecture can construct meta-memes which are stored in the selection mechanism, allowing higher-order learning to occur automatically.

6 Traveling salesman problem

The Traveling Salesman Problem (TSP) is a standard combinatorial optimization problem used for the design and evaluation of optimization methods [3,5,7,11,37,44,46,47,65,66,69,72,74]. TSP optimization algorithms have a wide range of applications including job scheduling, DNA sequencing, traffic management, and robotic path planning. To further illustrate the capabilities of the meta-learning design paradigm, an example is presented using instances of the TSP.

To apply meta-learning to the TSP problem, the schema of the problem must be identified. Here the schema takes the form of the ordering of points in a tour. The addition of a clustering method to divide and conquer the TSP has been shown to greatly accelerate the solution of the TSP [40]. With this addition, the overall schema for the optimizer consists of the combination of cluster templates, tour point ordering, and the locations of points. This schema must be generalized to create a meme, which is trivial for the cluster templates, but more challenging for the tour ordering and points' locations. The problem is further complicated by the necessity to generalize tours to be applicable over multiple scales.

For this application, a meme consists of a small ordered tour, containing small, limited number of points. To generalize the meme, the centroid of the group is calculated and subtracted from each point, making the centroid the origin of the group. The coordinates of each point are then normalized by distance from the origin. This projects the points into unit-space, and allows comparisons across multiple scales. Each TSP-meme serves as a pre-optimized tour template. Each point in the TSP-meme can represent a real point in the problem instance, or the centroid of a group of points, itself represented by a meme.

Given an instance of the TSP, the meta-TSP algorithm utilizes a clustering method to divide the problem into sub-problems, and divides those sub-problems into sub-sub problems and so on, until a threshold for sub-problem size is reached. The relationships between sub-problems are recorded in a tree-representation. Each of these sub-problems is generalized, and compared against the recorded memes for existing solutions.

The recognition mechanism must be able to detect structurally similar sub-problems. For this experiment the matching mechanism compares two normalized sub-problems by

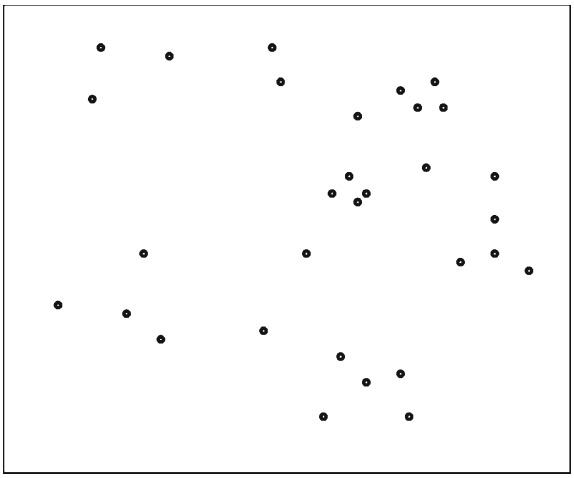

Fig. 12 Small TSP instance of approximately 30 points

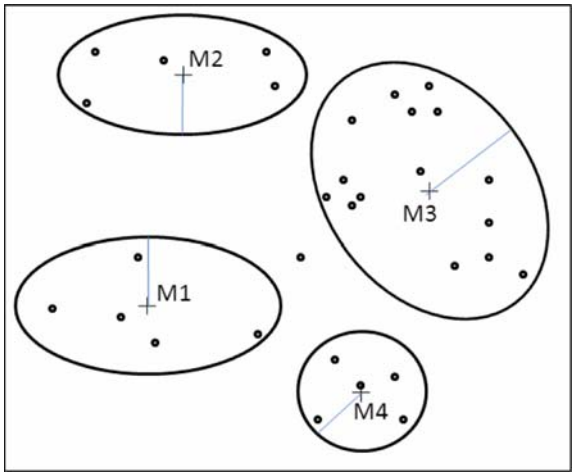

Fig. 13 TSP Instance after first clustering pass. Each cluster initializes a meme, labeled with 'M#' and a '+' denoting the centroid

finding the nearest corresponding points between the memes, and calculating the mean squared error between these points.

If a match is found in memory, the existing meme-solution (a point ordering) is copied to the current sub-problem, and the sub-problem updates the meme by refining template point positions. If no match exists in memory, the sub-problem is solved as accurately as possible. With a small enough problem threshold, exact solutions to sub-problems can be found, depending on computational resources available. The sub-problem is then stored in memory as a new meme. After all the sub-problems are solved, they are combined into a global tour by collapsing the problem-tree, and utilizing a simple $O(n)$ merge algorithm as detailed in Mulder and Wunsch [44].

To illustrate this process, an example is given utilizing a simple instance of the TSP, shown in Fig. 12. A first pass of clustering is shown in Fig. 13. Note that cluster M3 contains many points, and that a single point has been left out of the clusters for illustrative purposes. A second pass further divides cluster M3 into clusters M5, M6, and M7, as shown in Fig. 14. The final clustering pass assigns all clusters to a global cluster, M8, in Fig. 15. The hierarchy of clusters, and thereby sub-problems, is denoted by the cluster tree in Fig. 16.

At this stage, each sub-problem is optimized independently, as shown in Fig. 17. Note that some of the sub-problems contain references to other sub-problems, particularly M3 and M8. The centroids of sub-problems are utilized for optimization and solution, representing sub-problems as a whole. During the course of optimization, each sub-problem is normalized, and compared with previously computed, normalized solutions in the memory. These memes can be stored across instances, building a large library of pre-computed solutions that can be deployed to yield high quality solutions rapidly. Sub-problems of a global problem instance

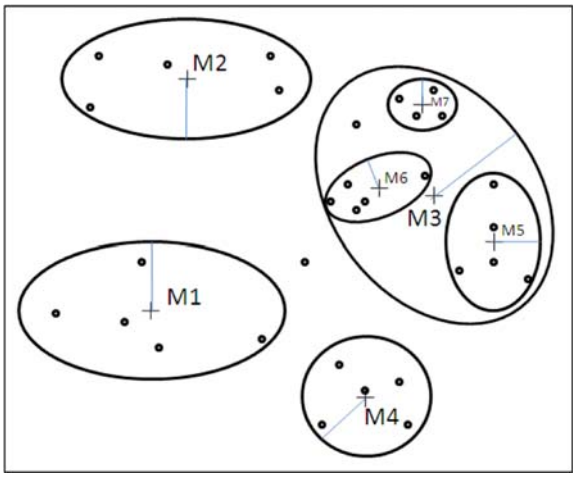

Fig. 14 Second clustering pass. Note the new clusters, M5, M6, and M7

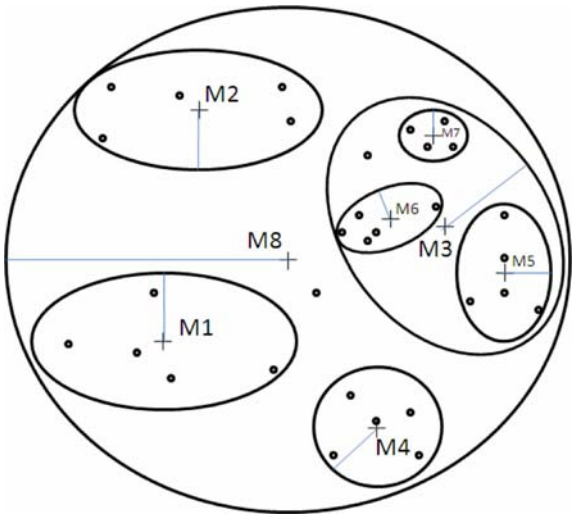

Fig. 15 Final clustering pass, with global cluster M8

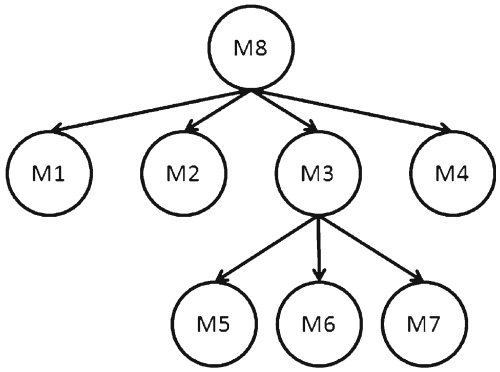

Fig. 16 Tree of sub-problems (clusters)

can be thought of as new problem instances, and pre-computed solutions that are generated during the calculation of a global instance can be applied across sub-problems.

For example, the normalized versions of M2 and M4 would be very similar in structure, and once M2 is computed, the structural similarity of the sub-problems would be recognized, and the ordering of points for M4 need not to be computed, only copied from M2 to M4. The same process applies across scales and global problem instances.

When all sub-problems are completed, the problem hierarchy is collapsed by de-referencing sub-problems and incrementally merging them with higher level tours. This is accomplished by choosing the closest set of two vertices in the sub-problem to any two vertices in the higher level tour. To avoid an $O(n^2)$ operation, a small neighborhood of vertices from the super-tour is chosen based on proximity to the centroid of the sub-tour. This neighborhood of super-tour vertices is compared to each vertex in the sub-tour to find the best match. A result of this merge operation is illustrated in Figs. 18 and 19. Figure 19 shows the final merge of all complete sub-tours into a final tour. The completed tour is shown in Fig. 20.

The computational complexity of the proposed method is expected to be very efficient at $O(n \log(n))$ improving with linearly decreasing complexity as the library of pre-optimized solutions grows, decreasing the amount of optimization to be performed on a given TSP instance.

The qualitative performance of this method is the subject of future development. The algorithm presented here serves as an example of meta-learning driven design, incorporating mechanisms of memory, selection, optimization, and generalization.

7 Conclusion

The desire for a new and robust computational intelligence paradigm spans many problem domains, including real time robotic systems which must deal with increasing complexity on a daily basis, deep data mining such as natural language processing with applications in information retrieval and machine understanding, human–computer interaction,

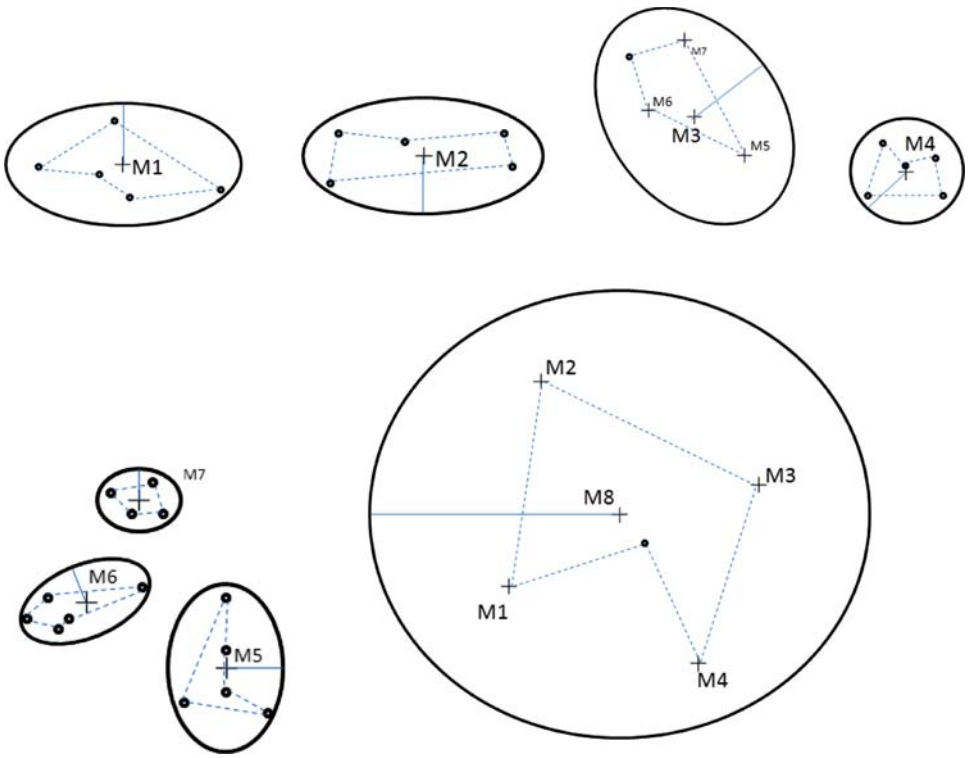

Fig. 17 Completed memes, M1 through M8. Super-clusters reference the centroids of sub-clusters. Note that memes M2 and M4 are similar in structure, but not scale

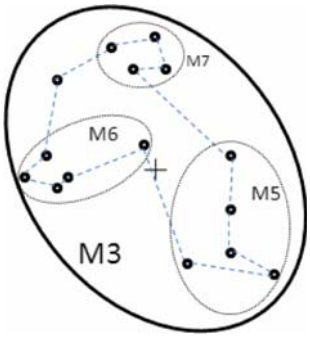

Fig. 18 Memes M5, M6, and M7 are merged into M3

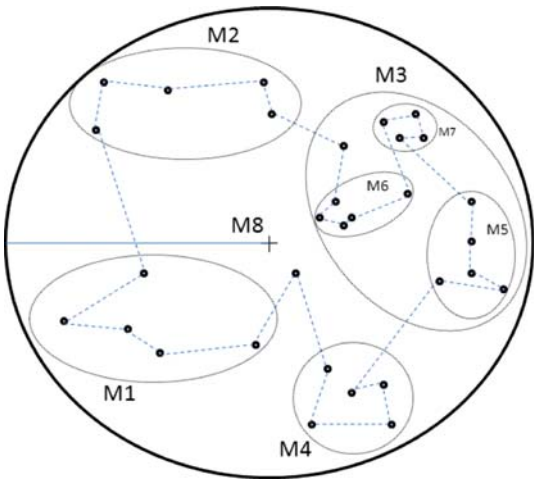

Fig. 19 Memes M1, M2, M3, M4 are merged into M8

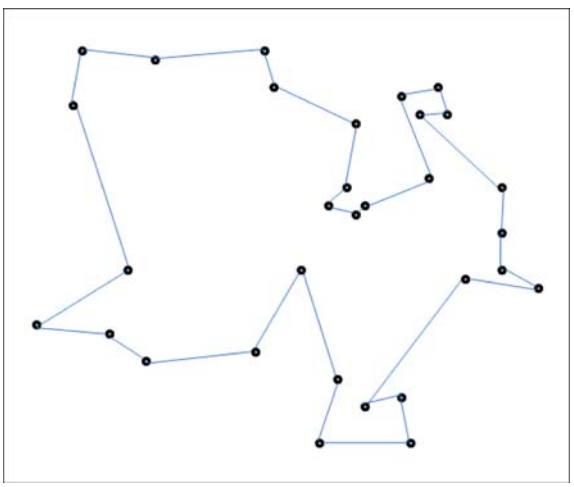

Fig. 20 Completed tour

and long-term optimization. These new, complex frontiers of machine learning and optimization could all benefit from the higher order memetic computing methods described here.

We have presented an overview of important definitions and architectures in memetic computing and have attempted to illustrate the power of next-generation memetic algorithms. The primary difficulty of designing meta-learning systems lies in the construction of valid scale-invariant representations which enable the simple construction of selection, generalization, and memory mechanisms. By providing generalization, memory, optimization, and selection mechanisms, a meta-learning architecture can operate on high-level features of a problem instance, selecting generalized solutions that have been used previously with high utility in the problem context. Utilizing these features, a system should be able to learn not only the solution to a problem, but learn *about* solving problems. Such a system may enable a quantum leap in the performance of real-world adaptive systems as they provide the central components of meta-adaptive systems to be constructed.

References

1. Abramson M, Wechsler H (2001) Competitive reinforcement learning for combinatorial problems. In: International joint conference on neural networks proceedings. IJCNN '01
2. Agarwal A, Lim M-H, Er M-J, Chew C-Y (2005) ACO for a new TSP in region coverage. IEEE/RSJ Int Conf Intel Robot Syst
3. Agarwal A, Lim M-H, Er MJ, Nguyen TN (2007) Rectilinear workspace partitioning for parallel coverage using multiple UAVs. Adv Robot 21(1)
4. Angeline PJ (1993) Evolutionary algorithms and emergent intelligence. Doctoral thesis, Ohio State University, Columbus
5. Applegate D, Cook W, Rohe A (2003) Chained Lin-Kernighan for large traveling salesman problems. INFORMS J Comput 15(1):82–92
6. Arnold DV, Salomon R (2007) Evolutionary gradient search revisited. IEEE Trans Evol Comput 11(4):480–495
7. Baraglia R, Hidalgo JI, Perego R (2001) A hybrid heuristic for the traveling salesman problem. IEEE Trans Evol Comput 5(6):613–622
8. Beinenstock EL, Cooper L, Munro P (1982) Theory for the development of neuron selectivity: orientation specifity and binocular interaction in the visual cortex. J Neurosci 2(1):32–48
9. Burke E, Cowling P, Causmaecker PD, Berghe G (2001) A memetic approach to the nurse rostering problem. Appl Int 15(3):199–214
10. Caponio A, Cascella GL, Neri F, Salvatore N, Sumner M (2007) A fast memetic algorithm for off-line and on-line control design of PMSM drives. IEEE Trans Syst Man Cybern Part B Spec Issue Memetic Algorithms 37(1):28–41
11. Dang J, Zhang Z (2005) A polynomial time evolutionary algorithm for the traveling salesman problem. Int Conf Neural Netw Brain
12. Dawkins R (1989) The selfish gene. Oxford University Press, USA
13. Francois O, Lavergne C (2001) Design of evolutionary algorithms-A statistical perspective. Evol Comput IEEE Trans on 5(2):129–148
14. Gaudiot J-L, Kang J-Y, Ro WW (2005) Techniques to improve performance beyond pipelining: superpipelining, superscalar, and VLIW. Adv Comput (63)
15. Gutin G, Karapetyan D (2009) A selection of useful theoretical tools for the design and analysis of optimization heuristics. Memetic Comput 1(1)
16. Hart WE (1994) Adaptive global optimization with local search. University of California, California

17. Hasan SMK, Sarker R, Essam D, Cornforth D (2008) Memetic algorithms for solving job-shop scheduling problems. Memetic Comput J
18. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
19. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans Evol Comput 7(2):204–223
20. Johansson C, Lansner A (2007) Towards cortex sized artificial neural systems. Neural Netw 20(1):48–61
21. Kazarlis SA, Papadakis SE, Theocharis JB, Petridis V (2001) Microgenetic algorithms as generalized hill-climbing operators for GA optimization. Evol Comput IEEE Trans on 5(3):204–217
22. Kendall G, Soubeiga E, Cowling P (2002) Choice function and random hyperheuristics. In: 4th Asia Pac Conf Simul Evol Learn, pp 667–671
23. Kolodner J (1993) Case-based reasoning. Morgan Kaufmann Publishers Inc., San Francisco
24. Koza JR (1989) Hierarchical genetic algorithms operating on populations of computer programs. In: International joint conference on artificial intelligence. Morgan Kaufman Publishers
25. Koza JR (1991) Evolution and co-evolution of computer programs to control independent-acting agents. In: From animals to animats: proceedings of the first international conference on simulation of adaptive behavior
26. Koza JR (1992) The genetic programming paradigm: genetically breeding populations of computer programs to solve problems. Dynamic, genetic and chaotic programming. Wiley, London, pp 201–321
27. Koza JR (1992) Hierarchical automatic function definition in genetic programming. Foundations of genetic algorithms, vol 2. Morgan Kaufmann, San Francisco, pp 297–318
28. Krasnogor N (2002) Studies on the theory and design space of memetic algorithms. PhD, Faculty Comput Math Eng Bristol, UK, University West of England
29. Krasnogor N, Blackburne B, Hirst JD, Burke EK (2002) Multimeme algorithms for the structure prediction and structure comparison of proteins. In: Proc. parallel problem solving from nature. Lecture notes in computer science. Springer, Heidelberg
30. Krasnogor N, Gustafson S (2004) A study on the use of self-generation in memetic algorithms. Nat Comput 3(1):53–76
31. Krasnogor N, Smith J (2005) A tutorial for competent memetic algorithms: model, taxonomy, and design issues. Evol Comput IEEE Trans 9(5):474–488
32. Kuncheva LI, Jain LC (2000) Designing classifier fusion systems by genetic algorithms. Evol Comput IEEE Trans 4(4):327–336
33. Land MWS (1998) Evolutionary algorithms with local search for combinatorial optimization. University of California, California
34. Lee JT, Lau E, Yu-Chi H (2001) The Witsenhausen counterexample: a hierarchical search approach for nonconvex optimization problems. Automat Control IEEE Trans 46(3):382–397
35. Lenat D, Guha RV (1989) Building large knowledge-based systems. Addison-Wesley, Reading
36. Lim M-H, Gustafson S, Krasnogor N, Ong Y-S (2009) Editorial to the first issue. Memetic Comput 1(1)
37. Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21(2):498–516
38. Merz P (2004) Advanced fitness landscape analysis and the performance of memetic algorithms. Evol Comput 12(3):303–325
39. Merz P, Freisleben B (1997) Genetic local search for the TSP: new results. IEEE Conf Evol Comput
40. Meuth RJ, Wunsch DC II (2008) Divide and conquer evolutionary Tsp solution for vehicle path planning. In: Congress on evolutionary computation (WCCI'08)
41. Milano M, Roli A (2004) MAGMA: a multiagent architecture for metaheuristics. Syst Man Cybern Part B IEEE Trans 34(2):925–941
42. Minsky M (1986) The society of mind. Simon & Schuster Inc, New York
43. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms, caltech concurrent computation program, C3P Report, 826
44. Mulder S, Wunsch DC (2003) Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. Neural Netw
45. Neri F, Toivanen J, Cascella GL, Ong Y (2007) An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE ACM Trans Comput Biol Bioinform 4(2):264
46. Nguyen HD, Yoshihara I, Yamamori K, Yasunaga M (2000) Modified edge recombination operators of genetic algorithms for the traveling salesman problem. In: 26th Annual conference of the IEEE industrial electronics society
47. Nguyen HD, Yoshihara I, Yamamori K, Yasunaga M (2007) Implementation of an effective hybrid GA for large scale traveling salesman problems. IEEE Trans Syst Man Cybern Part B 37(1):92–99
48. Nguyen Q-H, Ong Y-S, Lim M-H (2008) Non-genetic transmission of memes by diffusion. In: 10th Annual conference on genetic and evolutionary computation (GECCO'08), Atlanta, GA
49. Nguyen QH, Ong YS, Krasnogor N (2007) A study on the design issues of memetic algorithm IEEE congress on evolutionary computation singapore. IEEE 2390–2397
50. Norman MG, Moscato P (1989) A competitive and cooperative approach to comple combinatorial search, caltech concurrent computation program, C3P Report 790
51. O'Neill M, Ryan C (1999) Automatic generation of high level functions using evolutionary algorithms. In: 1st International workshop on soft computing applied to software engineering. Limerick University Press, Limerick
52. O'Neill M, Ryan C (2001) Grammatical evolution. IEEE Trans Evol Comput 5(4):349–358
53. Ong Y-S, Lim M-H, Zhu N, Wong K-W (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Trans Syst Man Cybern Part B 36(1)
54. Ong YS, Keane AJ (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Trans Evol Comput 8(2):99–110
55. Ong YS, Nair PB, Keane AJ (2003) Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA J 41(4):687–696
56. Ong YS, Nair PB, Lum KY (2006) Max-Min surrogate-assisted evolutionary algorithm for robust aerodynamic design. IEEE Trans Evol Comput 10(4):392–404
57. Poli R (2001) Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. Genet Program Evolvable Mach 2(2):123–163
58. Rosca JP (1995) Genetic programming exploratory power and the discovery of functions. Conference on Evolutionary Programming. MIT Press, Cambridge
59. Rumelhart DE (1980) Schemata: the building blocks of cognition. Theoretical issues in reading and comprehension. B. B. R.J. Sprio, & W.F. Brewer, Erlbaum
60. Shahaf D, Amir E (2007) Towards a theory of AI completeness. In: 8th Interational symposium on logic formalizations of commonsense reasoning
61. Smart W, Zhang M (2004) Applying online gradient descent search to genetic programming for object recognition. In: Second workshop on Australasian information security, data mining and web intelligence, and software internationalisation, Dunedin, New Zealand
62. Smith JE (2007) Coevolving memetic algorithms: a review and progress report. IEEE Trans Syst Man Cybern Part B 37(1):6–17

63. Tang J, Lim MH, Ong YS (2007) Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Comput 11(9):873–888
64. Topchy A, Punsch WF (2001) Faster genetic programming based on local gradient search of numeric leaf values. Genet Evol Comput Conf
65. Tsai H-K, Yang J-M, Kao C-Y (2002) Solving traveling salesman problems by combining global and local search mechanisms. Conf Evol Comput
66. Tsai H-K, Yang J-M, Kao C-Y (2004) An evolutionary algorithm for large traveling salesman problems. IEEE Trans Syst Man Cybern Part B 34(4):1718–1729
67. Ullah ASSMB, Sarker RA, Cornforth D, Lokan C (2007) An agent-based memetic algorithm (AMA) for solving constrained optimazation problems. In: IEEE congress on evolutionary computation, Singapore
68. Wang H, Wang D, Yang S (2009) A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. Soft Comput
69. Wang L, Maciejewski AA, Seigel HJ, Roychowdhury VP (1998) A comparitive study of five parallel genetic algorithms using the traveling salesman problem. In: First merged international conference and symposium on parallel and distributed processing
70. Wills LM, Kolodner J (1994) Towards more creative case-based design systems. In: Proceedings of the twelfth annual national conference on artificial intelligence (AAAI-94):50–55
71. Woldpert DH, Macready WG (1997) No free lunch theorms for optimization. IEEE Trans Evol Comput 1(1):67–82
72. Wunsch DC, Mulder S (2003) Using adaptive resonance theory and local optimization to divide and conquer large scale traveling salesman problems. Int Joint Conf Neural Netw
73. Xin Y (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
74. Xu R, Wunsch DII (2005) Survey of clustering algorithms. Neural Netw IEEE Trans 16(3):645–678