Preface

TOWARDS MORE EFFECTIVE METAHEURISTIC COMPUTING

1. INTRODUCTION

The engineering and business problems we face today have become more impenetrable and unstructured, making the design of a satisfactory algorithm nontrivial. Traditionally, researchers strive to formulate the problems as a mathematical model by relaxing, if necessary, hard objectives and constraints. The exact solution to this formulated version can be obtained through manipulations of mathematical programming such as integer linear programming, branch-and-bound and dynamic programming. However, due to the enumeration nature of these methods, mathematical programming techniques are limited to the applications with small problem size. As an alternative, approximation solutions are targeted by problem-specific heuristics which analyze the properties and structures of the underlying problems and greedily construct a feasible solution. The quality of the generated solution varies significantly upon problem instances, though the greedy heuristics are usually computationally fast. It has been a dilemma for the choice between the mathematical programming techniques and heuristic approaches. A notion of levering the two solution methods has created the regime of *metaheuristic* which was first coined by Fred Glover (1986). The metaheuristic approach guides the course of a heuristic to search beyond the local optimality by taking full advantage of strategic level problem solving using memory manipulations without a hassle to design problem-specific operations each time a new application shows up and still inheriting the computational efficiency from the embedded heuristic. Metaheuristic has built its foundations on multidiscipilary research findings ranging from phylogenetic evolution, sociocognition, gestalt psychology, social insects foraging, to strategic level problem solving. From a broader perspective, *nature-inspired metaheuristics* focusing on metaphors share its name with evolutionary algorithms, artificial immune systems, memetic algorithm, simulated annealing, ant colony optimization, particle swarm optimization, etc (Holland, 1975; Kirkpatrick et al., 1983; Dorigo, 1992; Kennedy & Eberhart, 1995). *Strategic level metaheuristics* incorporate higher level problem solving mechanisms from artificial intelligence relying on rules and memory. Typical renowned methods at least include tabu search, scatter search, GRASP, variable neighbourhood search (Glover, 1989; Laguna & Marti, 2003; Feo & Resende, 1995; Mladenovic & Hansen, 1997).

With technologies and conceptions emerging over the years, the development of metaheuristic has come to a new era. Researchers and practitioners intend to identify the primitive components contained in metaheuristics and try to develop the so-called hybrid metaheuristics towards more effective metaheuristic computing. In light of this, several innovations have been proposed under variable categories such as *Matheuristic, Hyper-heuristic*, and *Cyber-heuristic*. These methodologies do not stick to a particular metaheuristic method, instead, an abstract model is defined. A hybrid metaheuristic algorithm can be automatically constructed or evolved using the abstract model. These innovative ideas advance the research of metaheuristic computing into a new generation. Another desired result of the intensive research on fundamentals of metaheuristics is the provision of unified development frameworks for constructing various forms of metaheuristics. These frameworks are easy enough for practitioners to construct a main-stream metaheuristic program, and are also flexible for researchers to create a sophisticated metaheuristic algorithm.

The remainder of this chapter is organized as follows. Section 2 reviews principal metaheuristics according to the classification of nature-inspired computation vs. strategic level problem solving. In Section 3 we disclose the research trend in metaheuristics hybridizations. Section 4 presents the notion for establishing unified development frameworks for metaheuristics. Finally, conclusions are made in Section 5.

2. NATURE-INSPIRED COMPUTATION VS. STRATEGIC LEVEL PROBLEM-SOLVING

In this section we present the most important metaheuristics according to the nature-inspired computation vs. strategic level problem solving classification.

2.1 Nature-Inspired Metaheuristics

Metaheuristic algorithms based on natural metaphors are easy to describe and likely to catch the attention of wide audiences. Since the advent of evolutionary algorithms inspired by Darwinian Theory, the research of nature-inspired metaheuristics has grown at a fast speed with an overwhelming number of metaphors ranging from ants, termites, birds, fish, immune systems, bacteria, and jazz harmony, to most recent ones such as honey bees, fireflies, monkeys, and cuckoos. We briefly review some of the important ones in the following.

2.1.1 Evolutionary Algorithm

Evolutionary algorithms (EA) (Bäck, 1996) may be the earliest form of metaheuristics taking advantage of nature metaphors. The famous selection principle, survival of the fittest, from Darwinian Theory was easily introduced into computation field for evolving elite solutions of perplexing problems. The simplest form of EA proceeds as follows (see Figure 1). A population of random solutions is initiated as the gene pool. The generational cycle consisting of selection, crossover, and mutation is iterated to accomplish the genetic functions of selecting the fitter individuals for reproduction. Representative EAs have been recognized as genetic algorithm, evolutionary programming, genetic programming, and evolutionary strategy. A later form of EA called memetic algorithm embeds a local search procedure into the generational cycle such that the evolution is more directly guided towards fitter genes.

Figure 1. Summary of the EA conception

```
Initialize
   Generate the initial population at random
Repeat
   Evaluate the fitness of each individual
   Peform selection principle to create parent population
   Conduct crossover between the selected parents to produce offspring
        population
   Conduct mutation on the offspring
Until the stopping criterion is satisfied
Output the overall best individual
```

2.1.2 Ant Colony Optimization

The ant colony optimization (ACO) algorithm (Dorigo, 1992) is inspired by the research on the real ant behavior during food foraging. Ethologists observed that ants are able to construct the shortest path from their colony to the feeding source through the use of pheromone trails as follows. An ant leaves some quantities of pheromone as it walks and marks the path by a trail of this substance. The next ant will sniff the pheromone laid on different paths and choose the walking direction with a probability proportional to the amount of pheromone on it. The ant then traverses the chosen path and leaves its own pheromone. This is an autocatalytic (positive feedback) process which favors the shorter path along which more ants have previously traversed. ACO simulates this process to solve an optimization problem as summarized in Figure 2 by considering a better solution as a shorter path.

2.1.3 Particle Swarm Optimization

Kennedy and Eberhart (1995) gave the first particle swarm optimization (PSO) proposal for continuous function optimization. The theory on which the PSO is founded is *sociocognition*, which states that social cognition happens in the interactions among individuals in such a manner that each individual learns from its neighbors' behavioral models/patterns, especially from those learning experiences that are rewarded. Cognition emerges from the convergence of individuals' beliefs. PSO is also biologically inspired, drawing on the observation that a swarm of birds (or particles, using the terminology of PSO) flock synchronously, change direction suddenly, scatter and regroup iteratively, and finally perch on a target. Each individual particle benefits from the experience of its own and that of the other particles of the swarm during the foraging process. This form of social intelligence not only increases the success rate for food foraging but also expedites the process. The baseline PSO proceeds as follows. Given an optimization problem, a swarm of particles representing candidate solutions is generated at random. Each particle iteratively moves in the solution space by reference to best experiences. In the baseline PSO, a particle remembers its best position visited so far and the best position observed overall by its neighbors. Figure 3 summarizes the process of a baseline PSO.

Figure 2. Summary of the ACO conception

```
Initialize
    Encode the solution space by a connected weighted graph
    Set initial pheromone for each edge
Repeat
    For each ant do
        Randomly select a starting node
        Repeat
            Move to the next node according to the node transition rule
        Until a solution is constructed
    For each edge do
        Update the pheromone intensity using the pheromone updating rule
Until the stopping criterion is satisfied
Output the overall best solution
```

2.2 Strategic Level Problem Solving Metaheuristics

The major difference between the strategic level problem solving metaheuristics and the nature-inspired metaheuristics is that the former takes a higher degree of memory utilization. With varying forms of memory structure, purposeful strategies can be devised to enhance the balance between intensification and diversification types of search. In this section we introduce a prevailing set of strategic level problem solving metaheuristics.

2.2.1 Iterative Local Search

Iterative local search (ILS) (Tounsi & Ouis, 2008) could be the simplest strategy to target the global optimum in a given search space. ILS conducts a single-point (trajectory) search strategy which iteratively performs local search at different starting solution chosen strategically. Figure 4 summarizes the

Figure 3. Summary of the PSO conception

```
Initialize
    Generate a swarm of particles at random
Repeat
    Evaluate the fitness of each particle
    Determine the personal best position visited so far by each particle
    Determine the best position visited so far by the particle's neighbors
    Update particles' positions by reference to best experiences
Until the stopping criterion is satisfied
Output the overall best solution
```

conception of ILS. The home base keeps track of the local search region of which the local optimum will be sought by a local search procedure. Once the local optimum is identified, it is compared to the current home base in order to decide the next region of interest according to an acceptance function. The acceptance function can be designed in variable forms, such as: (1) always accept the local optimum as the new home base, (2) accept the local optimum as the new home base if it's better than the current home base, and (3) conditionally accept the local optimum according to the Metropolitan criterion. The Restarting feature of the ILS is an important strategy that is the central idea adopted in recent strategic level problem solving metaheuristics.

2.2.2 Tabu Search

"Tabu" comes from Tongan, a language of Polynesia, where it indicates things that cannot be touched because they are sacred, and the word now means prohibition. Tabu search (TS) (Glover, 1989) forbids the reverse search to solutions already visited. TS starts with an initial solution configuration chosen at random, and then moves iteratively from one configuration to another until a given stopping criterion is satisfied. At each iteration, a set of candidate moves are considered and thus identify a neighborhood of the current configuration. When a move is taken this move is recorded in the tabu list and will not be reversed in next few iterations. Glover pointed out that there is no value in choosing a poor move except for avoiding a visited path being re-examined. Hence, all candidate moves are sorted in decreasing order according to the quality of their resulting solutions, and one could execute the best nontabu move or the best tabu move that meets the aspiration level and consider the resulting solution as the new current configuration. Upon termination, the best configuration visited overall is considered as the best solution. The conception of TS is illustrated in Figure 5.

2.2.3 Greedy Randomized Adaptive Search Procedure

Greedy randomized adaptive search procedure (GRASP) (Feo & Resende, 1995) is a restart or iterative search process, where each iteration consists of two phases: greedy randomized construction phase and local search phase (see Figure 6). The greedy randomized construction phase starts with an empty solution and incrementally constructs the solution by repeatedly adding an element selected from the restricted

Figure 4. Summary of the ILS conception

```
Initialize
    Generate an initial solution at random
    Set the home base optimum to the initial solution
Repeat
    Apply a local search on the current solution and return a local optimum
    Determine the new home base according to an acceptance function
    Generate a new solution based on the home base
Until the stopping criterion is satisfied
Output the overall best solution
```

candidate list (RCL) which stores the elements whose insertion would cause an increment cost under a quality restriction criterion α, where $\alpha = 0$ corresponds to a pure greedy selection (i.e., the element with the minimal increment cost is always selected), and $\alpha = 1$ is equivalent to a random solution construction (any element without destroying the solution feasibility can be selected). In the local search phase, the neighborhood of the solution obtained from the greedy randomized construction phase is investigated until a local minimum is found. The two phases are restarted repetitively until a stopping criterion is met. Upon termination, the best overall solution is kept as the final result.

3. METAHEURISTICS HYBRIDIZATIONS

The obvious advantage of hybridizing metaheuristics is the potential for enhancing the intensification/diversification synergy and improving the selection for regions to be explored in the course of the search. Often, hybrid algorithms exhibit better performance when solving complex problems, such as the optimization of difficult multimodal functions. These mechanisms are mainly combinations of using metaheuristics, low-level heuristics, mathematical programming techniques, direct search methods, etc.

Many successful metaheuristics hybridizations have been proposed. The capability in enhancing exploitation search has made direct search methods as popular alternatives to be combined with metaheuristics. Hedar and Fukushima (2006) proposes the Directed Tabu Search (DTS) method by employing the Nelder & Mead Method to intensify the search in the promising areas identified by the tabu search. They have shown the superiority of DTS over several TS variants on a set of benchmark test functions. Vaz and Vicente (2007) introduces a hybridization of pattern search (another form of direct search) with PSO. The pattern search is used to improve the best particle in the swarm with a length-decreasing mesh structure.

Another viable thinking is to create a hybridization by marrying metaheuristic methods with mathematical programming techniques. The motivation is that metaheuristics are good at exploring large space

Figure 5. Summary of the TS conception

```
Initialize
    Encode an initial solution into a configuration
    Set the tabu list (TL) to empty
Repeat
    Generate a candidate list (CL) of neighbors by applying the move
        function to the current configuration
    Sort the members contained in CL according to solution quality
    Choose from the CL the best nontabu neighbor or the best tabu
        neighbor that satisfies the aspiration criterion
    Replace the current configuration by the selected neighbor
    Tally the performed move into TL with a tabu tenure
Until the stopping criterion is satisfied
Output the overall best solution
```

Figure 6. Summary of the GRASP conception

```
Repeat
   //Greedy randomized construction phase//
   Repeat
      Establish an RCL which stores the solution elements whose insertion
      would cause an increment cost satisfying a quality restriction criterion $\alpha$
      Insert an element randomly selected from the RCL into the current partial
      solution
   Until a complete solution is obtained

   //Local search phase//
   Investigate the neighborhood of the solution obtained from the previous phase
       until a local minimum is found
Until the stopping criterion is satisfied
Output the overall best solution
```

and identifying promising regions or directions where the global optimum might exist and mathematical programming techniques are efficient and effective in solving structured problems with small search regions. Plateau et al. (2002) hybridized path relinking technique with the interior point method for solving 0–1 linear programs. It is seen in Backer et al. (2000) that the constraint programming is combined with tabu search to tackle the vehicle routing problem. Both references indicated the significant benefit of this model-based metaheuristic framework.

Multiple metaheuristics can cooperatively work in a framework to create synergism. Talbi and Bachelet (2006) proposes the COSEARCH approach which manages the cooperation of tabu search and a genetic algorithm for solving large scale instances of the quadratic assignment problem (QAP). Wang et al. (2007) enhances the diversification capability of PSO by setting the less fit attributes contained in the global best solution as tabu-active and repelling the particles from the tabu area. Shen et al. (2008) proposes an approach called HPSOTS which enables the PSO to leap over local optima by restraining the particle movement based on the use of tabu conditions.

Among the many possible plans for creating hybridizations, three emerging frameworks, namely, Matheuristics, Hyper-heuristics, and Cyber-heuristics, have attracted a broad attention from the research community. We highlight the features of each framework in the following sections.

3.1 Matheuristics

From the perspective of history, there seems to be a departure between the development of mathematical programming and metaheuristics. At one end, mathematical programming techniques (e.g., primal and dual forms, Lagrangean method, branch-and-bound, dynamic programming, linear/nonlinear programming, constraint programming, goal programming) can derive exact solutions of underlying problems; however, they are constrained to the applications that involve small or moderately sized problems. At the other end, metaheuristic techniques (e.g., TS, ACO, PSO, GRASP) trade the solution quality to run time, and thus they are able to report near-optimal solutions in acceptable times. Only few literature references have disclosed that the knowledge from mathematical programming can be integrated into metaheuristics, and few attempts have been intended to incorporate metaheuristic components into the mathematical programming framework. However, a notion of *mathematical programming aware metaheuristic* (matheuristic) proposed recently has shown that it may be beneficial to marry the two paradigms. On the one hand, the derived intermediate solutions by mathematical programming techniques can be used to help metaheuristics handle complex constraints and reduce the search space. On the other hand, the mechanism of metaheuristics, if appropriately embedded into the framework of mathematical programming, can expedite the process of locating promising regions (e.g., by deriving a better bound for cutting).

A type of matheuristic methodology is to decompose the given problem into several restricted subordinate problems, and to solve each subordinate problem by using a mathematical programming approach. This form of “meta-exact” method not only takes full advantage of mathematical programming to solve the subordinate problems to optimality but also relies on the diversification capability of metaheuristics to well explore the large problem space. In a similar vein, the composition can be done by using a mathematical programming model to define the neighborhood (search corridor) of the incumbent solution of focus to the mastering metaheuristic algorithm. Another type of matheuristic methodology is to employ metaheuristic scheme as the column generation and cut generation in the branch and cut framework. The metaheuristic techniques are in general more efficient and effective in finding an initial feasible solution than classic mathematical programming approaches, thus having an opportunity to find a better branching variable and quality bound.

A number of successful applications for Matheuristics have been reported. Glover and Hanafi (2010) exploits the advantages of using metaheuristic methods as supplements to classical mixed integer programming (MIP) approaches. It indicates the ways for generating inequalities in reference to target solutions and target objectives to supplement the basic functions of metaheuristic methods as evidenced by the experience of applying classical MIP approaches. It also focuses on supplementary linear programming model embedding these inequalities in special intensification and diversification processes and proposes more advanced approaches for generating the target objectives. Plateau et al. (2002) hybridized interior point processes and metaheuristics for solving 0–1 linear programs. The interior points generated are combined by a path relinking template. Computational experiments are reported on 0–1 multiconstraint knapsack problems. Backer et al. (2000) combines constraint programming with several meta-heuristics ranging from a simple tabu search to guided local search and applies the technique to vehicle routing problems. The experimental result manifests significant benefit. Mostaghim et al. (2006) combines linear programming technique with PSO to solve continuous optimization with linear constraints. They first compute the basic feasible solutions through linear programming and use these solutions as the initial swarm of particles. A random set of feasible solutions (particles) are used to update the particles so that they never get infeasible. Talbi (2002) proposes a taxonomy of hybrid metaheuristics which also claims an emerging trend in developing a generic hybrid that combines exact algorithms with metaheuristics. Croce et al. (2011) presents a matheuristic procedure based on integer programming formulation to deal with the total completion time 2-machines flow shop problem. Puchinger and Raidl (2005) compiles a comprehensive survey discussing different state-of-the-art approaches of combining mathematical programming techniques and metaheuristics to solve combinatorial optimization problems. It introduced several works disclosing the usefulness and strong potential of the research direction of matheuristics.

3.2 Hyper-Heuristic

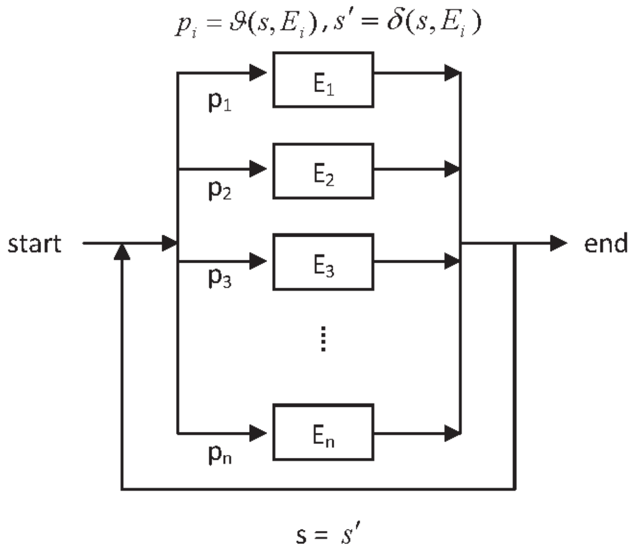
Denzinger et al. first coined the term Hyper-heuristic in 1997. Hyper-heuristic is an abstract model that describes how to select, combine, or generate more primitive heuristics to create an effective solution method for the given type of problems (Cowling et al., 2001; Burke et al., 2003; Özcan et al., 2008). It is thus representing a class of algorithms instead of a single one. The abstract model will generate a metaheuristic that is able to adapt to the characteristics of the given problem class. In conception, a set of feasible heuristics (constructive or perturbative) are identified and implemented. Then a learning engine would repeatedly generate higher-level heuristics by selecting or combining the primitive ones. Each produced higher-level heuristic is evaluated by performing to solve a training class of problems. The performance evaluation is then used to update the selection or combination probabilities. As a result, Hyper-heuristic can be thought of a heuristic for choosing heuristics, and it works in the heuristics space as a contrary to metaheuristic which usually works in the solution space of the problem at hand.

Hyper-heuristic can be performed in either macro-level or micro-level. The macro-level Hyper-heuristic automatically selects or combines known heuristics that were already designed for solving the given class of problems, while the micro-level Hyper-heuristic generates a generic new heuristic by working on the space of primitive components of known heuristics. The Hyper-heuristic model can be implemented using machine learning approaches (e.g., C4.5 and reinforcement learning) or evolutionary computation techniques (e.g., genetic programming and tabu search). It is not necessary to know intensive knowledge about the problem domain for constructing a Hyper-heuristic because the domain knowledge is encapsulated in the selected or combined heuristics.

Figure 7 shows an abstract model using Hyper-heuristic. The process repeatedly selects a pre-implemented heuristic element E_i according to an estimate of probability $p_i = \vartheta(s, E_i)$ for performing element E_i at a given state s. The state is then changed to a new state by $s' = \delta(s, E_i)$ as a consequence of the performed heuristics. The selection of the element can be practiced in various ways. The *first-best* scheme chooses the first element that leads to an improvement on the value of the estimate of merit. The *overall-best* scheme selects the element that gives the best value of the estimate of merit. *Probability* scheme associates each element with a selection probability based on the estimate of merit. *Random* scheme simply chooses an arbitrary element without biased probability. The *hybrid* scheme combines multiple alternatives above noted.

We have observed a number of successful applications for Hyper-heuristic. Cano-Belmán et al. (2010) proposes a scatter search based hyper-heuristic for addressing an assembly-line sequencing problem with work overload minimization. In the low-level, the procedure makes use of priority rules through a constructive procedure. In the strategic-level, the scatter search is used to select most suitable low-level heuristics to construct a new heuristics. Experimental results show that the proposed hyper-heuristics overcomes existing heuristics. Ouelhadj and Petrovic (2010) investigates the cooperation between low-level heuristics within a hyper-heuristic framework. Low-level heuristics perform local search through the same solution space, and the cooperative hyper-heuristic makes use of a pool of the solutions of the low-level heuristics for the overall selection of the low-level heuristics and the exchange of solutions. The experiment results show that the cooperative hyper-heuristics outperforms sequential hyper-heuristics. Burke et al. (2003) evaluates a hyper-heuristic approach on various instances of two distinct timetabling and rostering problems. The strategic-level hyper-heuristics selects low-level heuristics based on the reinforcement learning framework. A tabu list of heuristics is also maintained which pre-

Figure 7. An abstract model of Hyper-heuristic



vents certain heuristics from being chosen at certain times during the search. The proposed hyper-heuristics is capable of producing solutions that are competitive with those obtained using state-of-the-art problem-specific techniques. Özcan et al. (2010) proposes a hyper-heuristic framework which combines successive stages of heuristic selection and move acceptance. The heuristic selection is automatically controlled by the reinforcement learning technique, while the great deluge is employed to implement the acceptance function of the solution move.

3.3 Cyber-Heuristic

Before the mid-1990 most nature-inspired metaheuristics made use of metaphors for combining solutions. But today there is a major emphasis to generate new solutions using high-level problem-solving strategies. These strategies may involve complex neighborhood concepts, memory structure, and adaptive search prinicples, mainly drawn from the field of Tabu Search. The reason for this paradigm shift is that although natural metaphors are useful to describe the rationale of solution reproduction, it may create a biased perspective about preferred forms of methods and also cause valuable alternative strategies to be overlooked.

The aim of developing Cyber-heuristic algorithms is to marry strategic-level problem solving metaheuristic approaches with nature-inspired metaheuristics to create additional benefits that cannot be obtained using either one of them alone. The adjective "Cyber" emphasizes the connection between the nature-inspired metaheuristics and the intelligent learning provided by the Scatter Search /Path Relinking (SS/PR) components, which in turn draw upon the adaptive memory programming suite. Following this vein, a number of metaheuristics hybridizations can be developed such as Cyber-EA, Cyber-ACO, Cyber-PSO, etc.

Omran (2011) edited a special issue on Scatter Search and Path Relinking methods which addresses the contributions of SS/PR components for swarm methods. The contrasting features of the two parties are that SS/PR utilizes systematic solution construction rather than using randomization, and that SS/PR employs adaptive memory strategies to select influencial solutions instead of drawing metaphors from nature. This difference creates a potential to marry them together and achieve synergism. Several successful hybridizations of SS/PR and swarm methods were proposed in this special issue, searving a good reference for further extensions.

Yin et al. (2010) introduces the Cyber Swarm Algorithm which gives more substance to the PSO metaphors by adapting them to refer to social interactions among humans as well as interactions among lower organisms. The conception of the Cyber Swarm Algorithm is summarized as follows. In this algorithm, the social learning of a particle is not restricted to the interaction with the previous best of its neighbors but instead, the learning is involved with members from a dynamically maintained reference set, a notion from Scatter Search, containing the best solutions found overall in history by reference to quality and diversity. The use of dynamic social network can provide more fertile information in guidance than the static social network. Path relinking diversification strategies are introduced to locate quality solutions that are diverse in various ways from those seen in the normal search course.

4. UNIFIED DEVELOPMENT FRAMEWORK

As most metaheuristics and their hybrids are described by an abstract model, it would be beneficial to create a metaheuristics development framework to assist researchers and practitioners to develop a metaheuristic algorithm customized to their underlying problems. We observed a number of program callable libraries provided by relevant research centers and institutes. Primitive functions and operators for varying metaheuristics were implemented and the users only need to define their problem and construct the desired solution method.

Among others, GAlib (Wall, 1996) is a framework for developing genetic algorithms. MAFRA (Krasnogor & Smith, 2000) allows the users to insert a local search procedure into an evolutionary algorithm, resulting in a memetic algorithm. Hotframe (Fink, Voss, & Woodruff, 1999) is for implementing trajectory-based (single solution) evolutionary algorithms. iOpt (Voudouris et al. 2001) is a framework for genetic algorithm and extended hybrid approaches. CIlib (Cloete, Engelbrecht, & Pampar, 2008) is a callable library for constructing many swarm-based and evolutionary algorithms. Particle Swarm Center (http://www.particleswarm.info/Programs.html) contains a number of useful sources for obtaining variable versions of Particle Swarm Optimization. MetaYourHeuristic (http://intelligence.im.ncnu.edu.tw) can facilitate flexible course of metaheuristic construction, including GA, ACO, PSO, SA, TS, SS, GRASP, and Hyper-heuristics. ParadiseEO (http://paradiseo.gforge.inria.fr) can handle multiobjective optimization problems and KanGAL (http://www.iitk.ac.in/kangal/codes.shtml) provides source codes for single and multiobjective GAs.

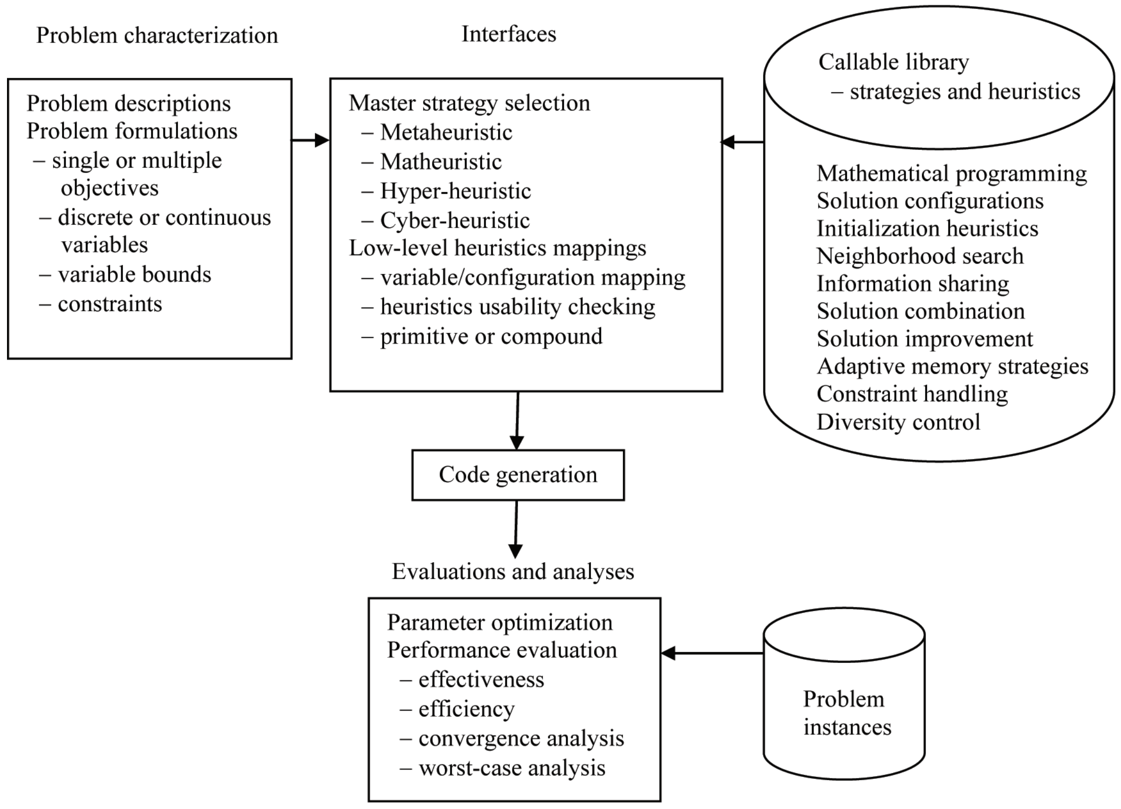
Yaghini et al. (2010) proposed a framework called DIMMA for standardizing the design, implementation, and evaluation of variable metaheuristic algorithms. From the perspective of software engineering, DIMMA divides the time axis of metaheuristics development into three phases, initialization, blueprint, and construction. Each phase requires various domains of disciplines, such as problem definition, heuristic strategies and components, parameter optimization, performance testing metrics, etc.

A general conceptual diagram for a unified development framework of metaheuristic algorithms can be depicted in Figure 8. There are four main components constituting this flexible framework. The *problem characterization* component allows the user to describe the addressed problem in a documentary and then to define the problem according to a formal definition language such as AMPL and GAMS. This could be nontrivial because the addressed problem may have more than one possible formulation and the merit of each formulation should be evaluated with the used (meta)heuristic method and the employed solution configuration, and one needs to conduct a design of experiment (DOE) approach in order to seek a good combination.

The *callable library* component contains computation procedures and operations that are frequently used in variable mathematical programming techniques and (meta)heuristic methods. Useful mathematical programming techniques include linear programming, branch and bound, dynamic programming, constraint programming, and the like. The common core procedures and operations used in many (meta) heuristics involve initialization heuristics, neighborhood search heuristics, solution selection, information sharing, solution combination, solution improvement, adaptive memory strategy, constraint handling, diversity control, among others. Moreover, the solution configuration representation is another matter of concern. The formulated problem is not restricted to a fixed solution configuration. The solution may appear in a form of array, tree, path, or any particular graph. Different solution configurations would affect the selection of applied (meta)heuristics.

The *interface* component plays a central role in this development framework. First, it separates the framework into three levels, namely, the problem abstract level, solution operation level, and system performance level. Each level is transparent to the others. In other words, the modification of codes in any level would not affect those in the rest of the framework. Second, the interface component encapsu-

Figure 8. Unified development framework of metaheuristic algorithms



lates high-level problem solving strategies which purposefully organize the procedures and operations called from the library. For instance, a Metaheuristic strategy will require the user to specify the values of appropriate parameters for calling the procedures and operations of the indicated method (GA, SA, TS, SS, etc.) A Matheuristic strategy may activate a problem decomposition technique and calls the linear programming procedure to solve the subordinate problem. Hyper-heuristic strategy usually needs to specify a heuristic selection method and a function to decide whether to accept the selected heuristic. The heuristic selection method ranges from random selection, performance proportional, rank truncation, to any metaheuristics and machine learning techniques. The criteria used by the acceptance function include accept all, accept on solution improvement, accept on Metropolitan criterion, to name a few. The Cyber-heuristic strategy should indicate the nature-inspired metaheuristic method as the inhabited and the embedding search strategies introduced in the Scatter Search and Path Relinking (SS/PR) template. Third, the interface component deals with the mapping between solution feasibility and operation applicability. As previously mentioned, there may exist more than one solution configuration form for a given problem formulation. Some configurations may be easier to handle the solution constraints and in essence reduce the search space. The low-level heuristic operations are usually dependent on the problem constraints and the solution configurations. The applicability of various heuristics should be taken into account in anticipation of the interface mapping selection. Another issue is the selection for primitive or compound operations for construction and perturbation of the solution at hand. There is yet no theoretic or empirical proof showing that either the compound operations are favorable or individual primitive operations are prevailing. Further, there could exist some correlations between the operations employed in the same framework. Some operations may be preferably performed in a sequence, but some operations could deteriorate the performance if they are bound in the same method. The optimization problems having similar properties in the objectives, variables, and constraints may benefit from similar patterns in adopting the heuristic operations. Finally, all viable interface mappings are evaluated using the DOE approach and the one with the best performance is realized.

The *evaluations and analyses* component provides a systematic way of deciding the final appropriate form of the established metaheuristic algorithm. One should evaluate and analyze the strengths and weaknesses of each mapping between interested problem formulations, solution configurations, and applied heuristics. For a tentative mapping, a sequence of experiments is advised. The parameter optimization experiment evaluates the mapping on an orthogonal set of parameter value combinations and analyzes the result with appropriate statistical tests. To obtain good generalization of the developed metaheuristic algorithm, a representative set of problem instances should be prepared and experimented with. The performance evaluation experiment then proceeds with the best parameter values identified in the previous phase and is conducted with a suite of performance metrics such as effectiveness, efficiency, convergence, and worst-case analyses. The effectiveness evaluation asserts the quality of the objective value obtained by the applied metaheuristics. One can compare the objective value obtained by competing metaheuristics given the same stopping criterion. Another popular measure for effectiveness is, the number of times a compared metaheuristic algorithm can produce a solution satisfying a target level. The efficiency analysis measures the computational time consumed by the metaheuristics to reach the target objective value. Since metaheuristics usually involve randomization, some statistics (mean, standard deviation, minimum, maximum, median) on the computational times across a larger number of repetitive nuns are favorable. The convergence of the tested metaheuristic algorithm can be analyzed by measuring the information gain during successions of iterations. As the number of iterations increases, the metaheuristic algorithm collects more information through exploration of the solution space and gradually resorts to some promising regions. The information entropy or variation metrics are suitable for analyzing the convergence. To provide a good Quality of Service (QoS), it is crucial to analyze the worst optimal performance we can obtain using the applied metaheuristics with various numbers of repetitive runs. As such different levels of quality guarantee can be facilitated.

More advanced schemes for selecting the appropriate mapping can be implemented based on self-adaption or meta-evolution. The space of variable mappings is explored using machine learning techniques or metaheuristic computing methods, while for each tentative mapping the principles of the implied method are used to guide the search for optimal solutions.

5. CONCLUSION

Many real-world problems are unstructured or semi-structured, making the design of a satisfactory algorithm nontrivial. Metaheuristic algorithms have emerged as viable solution methods for solving the problems to which the traditional mathematical programming and heuristics are ill-suited. The metaheuristic approach guides the course of a heuristics to search beyond the local optimality by taking full advantage of nature metaphores or problem-solving strategies. With our recent survey on metaheuristic algorithms, this chapter discloses two research trends that might make metaheuristics more effective. The first trend constitutes various proposals of metaheurisitcs hybridizations. Matheuristics, hyper-heuristics, and cyber-heuristics are notable among others. The second trend is the birth of many metaheuristics development frameworks. We present a unified development framework as a general conception for varying models. It would be beneficial for both practitioners and researchers to develop their metaheuristic algorithms by reference to this unified development model such that many possible plans for formulating problem, selecting search strategy, and performance evaluation can be systematically explored. The chapter has shown the great potentials of metaheuristic computing and invited more fruitful strategies from other domains to be injected into this ever exciting field.

Peng-Yeng Yin
National Chi Nan University

REFERENCES

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford Univ. Press.

Backer, B. D., Furnon, V., Shaw, P., Kilby, P., & Prosser, P. (2000). Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, *6*(4), 501–523. doi:10.1023/A:1009621410177

Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, *9*(6), 451–470. doi:10.1023/B:HEUR.0000012446.94732.b6

Cano-Belmán, J., Ríos-Mercado, R. Z., & Bautista, J. (2010). A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. *Journal of Heuristics*, *16*, 749–770. doi:10.1007/s10732-009-9118-2

Cloete, T., Engelbrecht, A. P., & Pampar, G. (2008). *CIlib: A collaborative framework for computational intelligence algorithms – part I*, IJCNN 2008. *IEEE World Congress on Computational Intelligence*.

Cowling, P., Kendall, G., & Soubeiga, E. (2001). A Hyper-heuristic approach to scheduling a sales summit. *Lecture Notes in Computer Science*, *2079*, 176–190. doi:10.1007/3-540-44629-X_11

Croce, F. D., Grosso, A., & Salassa, F. (2011). Lecture Notes in Computer Science: *Vol. 6622*. *A matheuristic approach for the total completion time two-machines permutation flow shop problem* (pp. 38–47).

Dorigo, M. (1992). *Optimization, learning, and natural algorithms*, Ph.D. Thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Italy.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*, 109–133. doi:10.1007/BF01096763

Fink, A., Voss, S., & Woodruff, D. L. (1999). Building reusable software components for heuristic search. In Kall, P., & Luthi, H. J. (Eds.), *Operations Research Proceedings* (pp. 210–219). Heidelberg: Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*, 533–549. doi:10.1016/0305-0548(86)90048-1

Glover, F. (1989). Tabu search - Part I, *ORSA J. Comput.*, *1*, 190–206.

Glover, F., & Hanafi, S. (2010). Metaheuristic search with inequalities and target objectives for mixed binary optimization part I: exploiting proximity. *International Journal of Applied Metaheuristic Computing*, *1*(1), 1–15. doi:10.4018/jamc.2010102601

Hedar, A., & Fukushima, M. (2006). Tabu search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research*, *170*(2), 329–349. doi:10.1016/j.ejor.2004.05.033

Holland, J. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence* (pp. 175–177). Ann Arbor: University of Michigan.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks, IV,* 1942-1948.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680. doi:10.1126/science.220.4598.671

Krasnogor, N., & Smith, J. (2000). MAFRA: a Java memetic algorithms framework. In Freitas, A. A., Hart, W., Krasnogor, N., & Smith, J. (Eds.), *Data Mining with Evolutionary Algorithms* (pp. 125–131). Las Vegas, NV.

Laguna, M., & Marti, R. (2003). *Scatter Search*. Boston: Kluwer Academic Publishers.

Mladenovic, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097–1100. doi:10.1016/S0305-0548(97)00031-2

Mostaghim, S., Halter, W. E., & Wille, A. (2006). Linear multi-objective particle swarm optimization, *Stigmergy optimization*, 31, 209-237.

Omran, M. (2011). Special Issue on Scatter Search and Path Relinking Methods. *International Journal of Swarm Intelligence Research*, *2*(2).

Ouelhadj, D., & Petrovic, S. (2010). A cooperative hyper-heuristic search framework. *Journal of Heuristics*, *16*, 835–857. doi:10.1007/s10732-009-9122-6

Özcan, E., Bilgin, B., & Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, *12*(1), 3–23.

Özcan, E., Misir, M., Ochoa, G., & Burke, E. K. (2010). A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, *1*(1), 39–59. doi:10.4018/jamc.2010102603

Plateau, A., Tachat, D., & Tolla, P. (2002). A hybrid search combining interior point methods and metaheuristics for 0-1 programming. *International Transactions in Operational Research*, *9*(6), 731–746. doi:10.1111/1475-3995.00385

Puchinger, J., & Raidl, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. *Lecture Notes in Computer Science*, *3562*, 41–53. doi:10.1007/11499305_5

Shen, Q., Shi, W. M., & Kong, W. (2008). Hybrid particle swarm optimization and tabu search approach for selecting genes for tumor classification using gene expression data. *Computational Biology and Chemistry*, *32*(1), 52–59. doi:10.1016/j.compbiolchem.2007.10.001

Talbi, E. G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, *8*(5), 541–564. doi:10.1023/A:1016540724870

Talbi, E. G., & Bachelet, V. (2006). COSEARCH: a parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, *5*(1), 5–22. doi:10.1007/s10852-005-9029-7

Tounsi, M., & Ouis, S. (2008). An iterative local-search framework for solving constraint satisfaction problem. *Proceedings of Appl. Soft Computing*, *n.d.*, 1530–1535.

Vaz, A. I. F., & Vicente, L. N. (2007). A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, *39*, 197–219. doi:10.1007/s10898-007-9133-5

Voudouris, C., Dorne, R., Lesaint, D., & Liret, A. (2001). iOpt: a software toolkit for heuristic search methods, In *CP'01 7th International Conference on Principles and Practice of Constraint Programming*: LNCS Vol. 2239 (pp. 716–729), Springer.

Wall, M. (1996). *GAlib: A C++ library of genetic algorithm components (Technical Report)*. USA: Mechanical Engineering Department, Massachusetts Institute of Technology.

Yaghini, M., & Kazemzadeh, M. (2010). DIMMA: a design and implementation methodology for metaheuristic algorithms – a perspective from software development. *International Journal of Applied Metaheuristic Computing*, *1*(4), 57–74. doi:10.4018/jamc.2010100104

Yin, P. Y., Glover, F., Laguna, M., & Zhu, J. X. (2010). Cyber swarm algorithms – improving particle swarm optimization using adaptive memory strategies. *European Journal of Operational Research*, *201*(2), 377–389. doi:10.1016/j.ejor.2009.03.035