An Agent-Based Hyper-Heuristic Approach to Combinatorial Optimization Problems

Richard Malek

Department of Cybernetics, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
malekric@fel.cvut.cz

Abstract. This paper introduces a framework based on multi-agent system for solving problems of combinatorial optimization. The framework allows running various meta-heuristic algorithms simultaneously. By the collaboration of various meta-heuristics, we can achieve better results in more classes of problems. Our hyper-heuristic approach is defined as a high-level search in algorithm space implemented within agents.
Key words: combinatorial optimization, meta-heuristic algorithm, hybrid approach, hyper-heuristic, multi-agent system

1 Introduction

A lot of practical problems (circuit board design, production planning and scheduling, vehicle routing) can be reduced to problems of combinatorial optimization.

Combinatorial optimization is a process of finding one or more best (optimal) solutions in a well defined discrete problem space. Each possible solution has an associated cost. The goal is to find the solution (a configuration, a combination of values) with the lowest cost [1].

Classical combinatorial problems are: Traveling Salesman Problem (TSP), Boolean Satisfiability Problem (SAT) or Job Shop Scheduling Problem (JSSP) [2].

Unfortunately, most practical optimization problems are too complex (NP-complete or harder) and the only way how to get at least approximately optimal solutions is the use of probabilistic algorithms. Meta-heuristic algorithms are an important member of the group of probabilistic algorithms.

Meta-heuristic is a high-level strategy for solving optimization problem that guides other heuristics in a search for feasible solutions. Best known meta-heuristics are: Genetic Algorithms (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS) and many others [1] [3].

In this paper, every time we mention an algorithm we mean meta-heuristic algorithm.

According to the No-Free-Lunch theorem [4] there is no superior meta-heuristic feasible for all optimization problems or even for all instances within

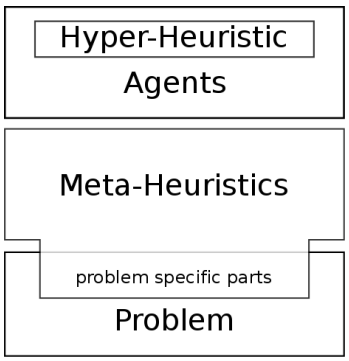
one class of problems. Recent developments in search methodologies towards more generally applicable techniques has been termed *hyper-heuristics* [5].

Hyper-heuristics are "heuristics to choose heuristics" [5]. They operate at higher level of abstraction than meta-heuristics and use particular (meta-)heuristics as building blocks of the optimization process. The majority of current hyper-heuristic approaches attempt to intelligently combine or select between previously proposed simpler heuristics, where it is not clear which one will be most effective for the problem instance at hand. Hyper-heuristics explore a search space of heuristics instead of a search space of solutions as meta-heuristics do [6].

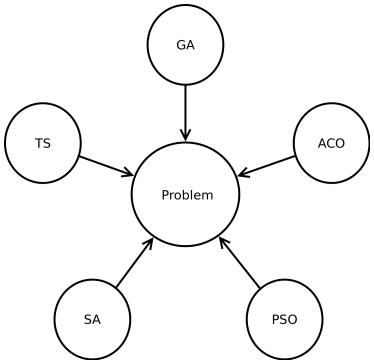
1.1 Our Approach

Our *hyper-heuristic* approach for solving *problems* of combinatorial optimization is provided as a repository of (partially) problem specific *meta-heuristics* (often referred to as 'low-level' meta-heuristics) and 'high-level' logic for their collaboration–implemented in *agents* within the multi-agent system (see Fig. 1(a)).

A particular problem is solved by various meta-heuristic algorithms altogether (see Fig. 1(b)). We use meta-heuristics implemented within the SEAGE [7] project.



(a) The structure view on our agent-based approach.



(b) Various meta-heuristics solving a problem simultaneously.

Fig. 1.

1.2 Related Work

MAGMA [8][9] is a MultiAGent Meta-heuristics Architecture conceived as a conceptual and practical framework for meta-heuristic algorithms. Authors revisit meta-heuristics in a multi-agent perspective and define agents of a different level

of abstraction. Level-0 agents provide feasible solutions for upper levels; Level-1 agents deal with solution improvements and provide local search; Level-2 agents have global view of the search space, or, at least, their task is to guide the search towards promising regions; Level-3 is introduced to describe higher level strategies like cooperative search and any other combination of meta-heuristics.

In [10], authors propose an agent-based multi-level search framework for the asynchronous cooperation of hyper-heuristics. This framework contains a population of different hyper-heuristic agents and a coordinator agent. Each hyper-heuristic agent operates on the same set of low level heuristics, while the coordinator agent operates on top of all the hyper-heuristic agents. Starting from the same initial solution, each hyper-heuristic agent performs a search over the space generated by the low level heuristics. The hyper-heuristic agents cooperate asynchronously through the coordinator agent by exchanging their elite solutions. The coordinator agent maintains a pool of elite solutions and manages the communication between the hyper-heuristics agents.

2 Approach Background

2.1 Different Meta-Heuristics, Different Points of View

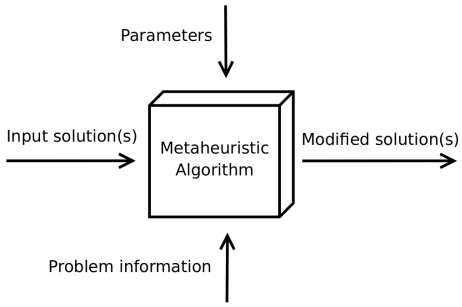
Different meta-heuristics have different inner representations of the problem they solve and objective functions (functions assigning the quality to a candidate solution) as well. Thus, different meta-heuristics differ in models of the solution space they seek through and have their own points of view on the problem. Watching the problem from different points of view could be, as we believe, the main advantage of the hyper-heuristic approach.

The use of different algorithms with their inherently different interfaces requires some kind of abstraction.

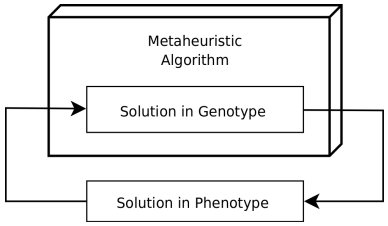
2.2 Abstraction of Meta-Heuristic Algorithm

In this section we will describe the interface all meta-heuristics can be handled through. This interface unifies an access to all meta-heuristics. We apprehend the meta-heuristic algorithm as a black box defined by its inputs and outputs. Each algorithm has to be initiated and its parameters have to be set as well. Then it takes candidate solutions on input, modifies them in a particular way, and puts the solutions on output. See Figure 2(a).

The inner representation of the candidate solution is the first thing we have to deal with during the algorithm unifying process. Such a representation is algorithm-specific. Dealing with various meta-heuristics, their different solution representations have to be mutually transformed. However, each problem can be described by algorithm-independent information. When we use a term from genetics, a solution of a given problem can be described by its phenotype [11][12]. It is a general description of the solution–the form that can be transformed into any other algorithm-specific representations.



(a) An abstraction of meta-heuristic algorithm.



(b) Algorithm converts solution from/into phenotype and genotype representation.

Fig. 2.

Thus, according to our abstraction, an algorithm takes the candidate solutions in phenotype representation on its input. It converts the solution into its inner representation–genotype. After processing, the solution is converted back to the phenotype representation and put on the output. See Figure 2(b). This evolves chaining of any algorithms that can do the phenotype/genotype mapping.

2.3 Population of Meta-Heuristic Algorithms

Once we have unified a view on different meta-heuristic algorithms we are able to use them all in process of searching for an optimal solution. Then we have two options how to run the algorithms. In a sequence or simultaneously. In both cases we have to select a feasible subset of all available algorithms and in case of sequencing even a proper algorithm order.

2.4 Population of Candidate Solutions

In our concept, the results of work of particular meta-heuristics are shared through a global population of candidate solutions. We denote the place, where the population is located, as a solution pool. The candidate solutions are loaded from and stored in the solution pool. Each meta-heuristic takes one or more solutions from and after it finishes, puts them back into solution pool. Simply, candidate solutions are improved by many different meta-heuristics and population of candidate solutions is being evolved.

3 Agent Approach Description

3.1 The Use of Multi-Agent System

Recently, multi-agent systems (MAS) have been used for solving a wide scale of artificial intelligence problems. For our purposes, the use of multi-agent system is very feasible as well. By using A-Globe [13] multi-agent system framework, we gained a lot of functionality for free (agent skeletons, message passing, agent addressing, conversation protocols, etc.).

According to our concept (presented in previous section), the system can be immediately decomposed into blocks such as particular meta-heuristics, (global) solution pool and others. Each block is handled by its agent. There are following agent types in our system: a problem agent, a solution pool agent, an algorithm agent and an adviser agent.

Problem agent is the entry point of our system. It can receive a request for the optimization task either from the user or from an agent of other system. The current implementation of the agent reads problem data from a configuration file. The agent initializes all other agents by sending the *init* message. It can receive two types of messages:

- *request* messages for initial solutions (the agent can generate initial solution since it has information about the problem)
- *inform* message when new best solution is found by the algorithm agent.

The agent also measures the overall optimization time and after timeout it sends *done* message to all agents and they finalize their work.

SolutionPool agent manages the population of candidate solutions and provides solutions to all algorithm agents. First, it asks problem agent for an initial population, then it receives messages from algorithm agents with requests either for loading or storing solutions. It can initialize itself after receiving *init* message as well.

Algorithm agent disposes of a particular meta-heuristic algorithm. It is responsible for:

- obtaining the algorithm parameter settings from the adviser agent
- asking the solution pool agent for candidate solution(s)
- running the meta-heuristic algorithm with received parameters and solution(s)
- sending the best solution found to the problem agent after the meta-heuristic algorithm is finished
- sending the solutions back to the solution pool agent
- sending a report on previous algorithm's run to the adviser agent

These actions are performed until *done* message is received from the problem agent.

Adviser agent agent provides parameter settings for the meta-heuristic algorithms and receives reports on algorithms' runs. All algorithm agents of the same class (operating with the same algorithm) have one corresponding Adviser agent associated. So, if there are five Algorithm agents–three with Genetic Algorithm, one with Tabu Search and one with Simulated Annealing algorithm, there are three Adviser agents–one for each algorithm class.

3.2 Agent Messaging

Figure 3 shows a system configuration with six agents. Arrows represent the agent communication (will be describe in next section).

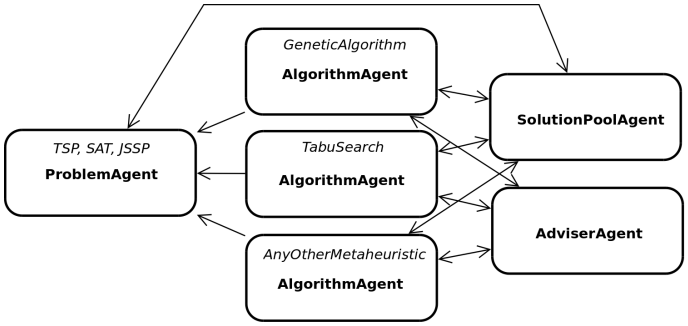


Fig. 3. Agents solving a problem. A configuration example.

Figure 4 shows a typical sequence of agent conversations.

1. The problem agent reads a configuration file with the information about agents it has to create and initializes them.
2. The solution pool agent asks the problem agent for the initial population and it sends the generated population of candidates solution back.
3. Once the algorithm agent is initialized (anytime), it asks the adviser agent for meta-heuristic parameters.
4. According to the obtained parameters, the algorithm agent asks the solution pool agent for solutions to work with and runs its meta-heuristic algorithm.
5. After finishing, the algorithm agent sends the best solution to the problem agent
6. The algorithm agent sends solutions back to the solution pool agent.
7. The algorithm agent also sends a search report to the adviser agent and it continues by the step 3.
8. After timeout the problem agent sends *done* message to all agents

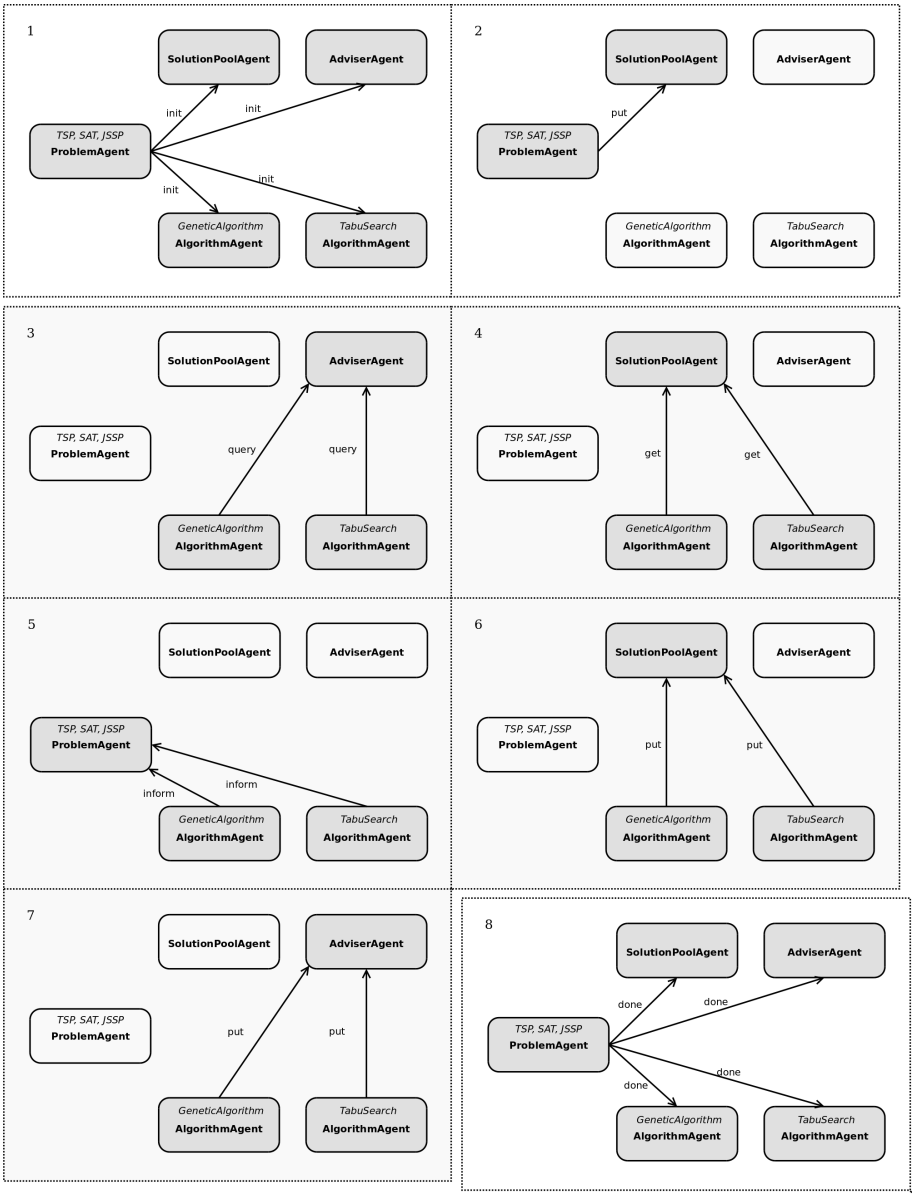


Fig. 4. A typical sequence of agent conversations. Actions in frames 3-7 are performed repeatedly.

4 Hyper-Heuristic Approach

4.1 Searching a Space of Meta-Heuristics

Hyper-heuristic approaches operate on a search space of heuristics rather than directly on a search space of solutions to the underlying problem–which is the case with most meta-heuristics implementations. The motivation behind hyper-heuristics is to raise the level of generality at which search methodologies operate [6].

In our approach we consider three kinds of algorithm spaces: the algorithm subsets space, the algorithm invocation sequence space and the space of algorithm settings.

Algorithm Subsets Space An important (very well known) observation which guides much hyper-heuristic research is that different heuristics have different strengths and weaknesses. A key idea is to use such a subset of all available meta-heuristics that are the most feasible for solving a given optimization problem. Beside of the reason of algorithm feasibility, limited resources are another reason why to deal with selecting of proper algorithm subsets.

At the first stage of our research algorithm subsets are selected by hand but it will change in the future–the process of selecting will be automatized. Our idea is that algorithm agents compete with each other and only a given number of them can do their work. Metrics for algorithm comparison is based on on meta-data collected during algorithms' work.

Algorithm Invocation Sequence Space In section 2.3 we mentioned running meta-heuristic algorithms in a sequence. Then the space of all sequences is formed by all possible orders of algorithm invocations. The sequence could be longer than just the number of available algorithms which makes the sequencing even harder.

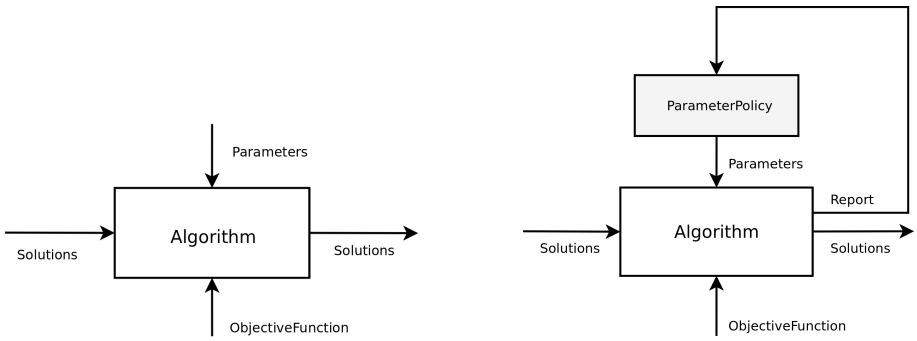
By using the multi-agent system, which is strictly parallel environment, we avoid the problem of sequencing. The sequence of particular algorithm applications inherently emerges by running all algorithms simultaneously.

Algorithm Settings Space It is commonly known that performance of meta-heuristic algorithms is strongly dependent on proper settings of their parameters. The settings of particular algorithms are often problem-dependent and only reliable way how to obtain parameter values is by performing experiments. To do it by hand is ineffective and uncomfortable and our intention is, again, to automate it. We will introduce this concept in details.

4.2 Automated Parameter Settings

Policy First of all, we added an object called Policy to each algorithm which is responsible for parameter settings. The term Policy has been adopted from the

reinforcement learning branch [14] and it has similar meaning as well. It introduces a feedback to the algorithm. The Policy object takes report on previous algorithm's run on input and provide a set of new parameter values on output. See Figure 5(b). In case of the first algorithm's run there is no previous run and any report, default parameter values are set.



(a) Algorithm abstraction model.

(b) Algorithm abstraction model with a parameter policy.

Fig. 5.

Static vs. Dynamic Parameters Once we have introduced the Policy object we may distinguish two types of parameter settings:

- **static**: parameter values given by the Policy object are the same for all the time the algorithm is running.
- **dynamic**: parameter values given by the Policy object are dependent on the report on the previous algorithm's run. See Figure 5(b).

The main motivation for introducing the Policy object is providing dynamic parameter settings. This should improve algorithms' performance.

Policy Types

Dummy Policy is the simplest policy. It just provides always the same parameter values it was configured for. It can be consider as a provider of static parameters.

Random Policy is a dummy policy as well and it provides random parameter values. It is the simplest provider of dynamic parameters.

Rule-based Policy contains pre-generated IF-THEN rules that produces desired parameter values.

Neural Network Policy contains pre-generated neural network that produces desired parameter values.

See the right part of Figure 6.

Policy Builders For static parameter policies Dummy policy builders have been designed. They simply creates dummy or random policies. It is obvious that dynamic parameter policies have to be properly initialized to provide meaningful values. Or even better they should evolve themselves in time with the increasing number of algorithms' runs.

Our approach involves evolution-based (smart) strategies for preparing particular policies. We called the strategies as policy builders. See left part of the Fig. 6.

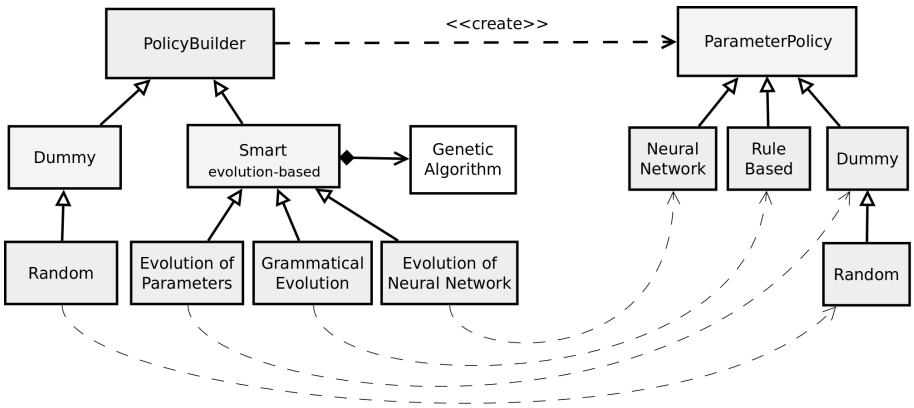


Fig. 6. Policy and policy builder hierarchy. The UML notation is used.

Evolution-based policy builders operate with a genetic algorithm and evolve a population of subjects that are represented as particular policies of following types:

Policy Builder with Parameter Values Evolution Subjects evolving by this builder represent directly parameter values.

Policy Builder with Grammatical Evolution This builder holds a grammar for generating IF-THEN rules. Subjects evolving by this builder then contain numbers of grammar rules to be applied to create particular rules.

Policy Builder with Neural Network Evolution Subjects evolving by this builder represent configuration and settings of neural networks.

Figure 6 shows how particular policies are created by particular policy builders (tiny dashed arrows).

Figure 7 shows integration policies and policy builders into agents. Algorithm agent uses parameter policy and Adviser agent contains a policy builder and a report evaluator.

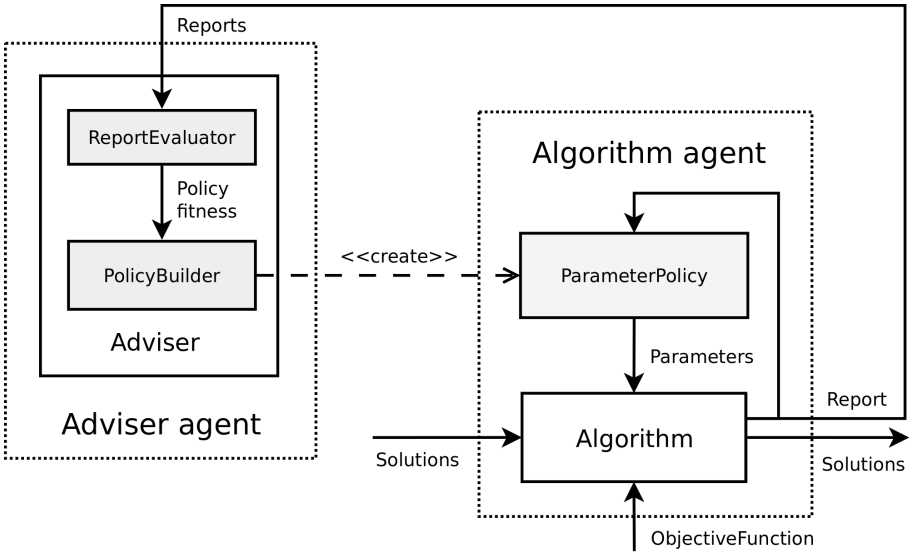


Fig. 7. A detailed view into Adviser and Algorithm agents.

Description of Evolution in Adviser In general, an evolution in Adviser agents is done as follows: There is an instance of Genetic Algorithm with an initial population. Operators of crossover and mutation are defined. When an Algorithm agent asks for an advise, a subject from population is selected, transformed into a Policy and send back to the Algorithm agent. Once the Algorithm agent finishes its job it creates a report and send it back to the Adviser agent (see Fig. 8). The report is evaluated by a Report evaluator (see Fig. 7) and an objective value (Policy fitness) is assigned to the subject.

After evaluating of all subjects in the population another evolution step is performed (application of genetic operators).

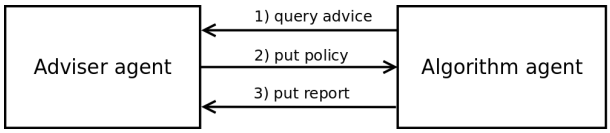


Fig. 8. A typical conversation between Algorithm and Adviser agents.

4.3 Meta-Data

Meta-data are information collected during algorithms' run. They contains a lot of run-time and statistical information and help to describe how an algorithm behaved during its run.

Here we provide some meta-data items we collect during each algorithm run-time:

- Algorithm-specific parameter values
  - The number of solutions
  - The number of iterations
  - ...
- New best ever solution records
  - Current iteration number
  - Current time
  - Solution's objective value
  - Solution value
- Run-time statistics
  - The number of new solutions found
  - The best of initial solutions' objective value
  - The best ever solution's objective value
  - The last improving iteration number
  - ...

Meta-data are contained in reports all algorithm agents provide. As mentioned above reports are evaluated and it enable to evolve new algorithm parameters.

5 Conclusions and Future Work

In this paper we have presented our concept for solving the problems of combinatorial optimization by employing various meta-heuristic algorithms. Being aware of No Free Lunch theorem, our concept is based on combination of particular meta-heuristics to achieve better results on wider scale of problems.

An approach of collaboration of different meta-heuristic algorithms through a multi-agent system has been introduced. To be such a implementation possible to realize, an abstraction of the meta-heuristic algorithm has been defined.

We found an architecture based on multi-agent system paradigm feasible for our purposes. Algorithm agents and the solution pool agent are the base of our concept of algorithm collaboration whereas the adviser agent is the base of our hyper-heuristic approach.

The core of our hyper-heuristic approach is in searching algorithms' parameter space and evolving parameter policies for dynamic parameter settings. We have presented several types of policies and principles of their evolution as well. Implementation and experiments are main subjects of future work.

References

1. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Frome, BA11 6TT, United Kingdom (2008)
2. Skiena, S.S.: The Algorithm Design Manual. Springer-Verlag (1998)
3. Yagiura, M., Ibaraki, T.: On metaheuristic algorithms for combinatorial optimization problems. Transactions of the Institute of Electronics, Information and Communication Engineers **J83-D-1**(1) 3–25
4. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
5. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology (2003)
6. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Woodward, J.R., Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E.: Exploring hyper-heuristic methodologies with genetic programming (2009)
7. Malek, R.: Seage – searching agents. http://www.seage.org
8. Roli, A., Milano, M.: Metaheuristics: a multiagent perspective. Technical report, University of Bologna (Italy (2001)
9. Roli, A., Milano, M.: Magma: A multiagent architecture for metaheuristics. IEEE Trans. on Systems, Man and Cybernetics - Part B **34** (2002) 2004
10. Ouelhadj, D., Petrovic, S., Ozcan, E.: A multi-level search framework for asynchronous cooperation of multiple hyper-heuristics. In: GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference, New York, NY, USA, ACM (2009) 2193–2196
11. Fogel, D.B.: Phenotypes, genotypes, and operators in evolutionary computation. (1995) 193–198
12. Back, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK, UK (1997)
13. Sislak, D., Rehak, M., Pechoucek, M.: A-globe: Multi-agent platform with advanced simulation and visualization support. Web Intelligence, IEEE / WIC / ACM International Conference (2005) 805–806
14. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (March 1998)