Collaboration of Metaheuristic Algorithms through a Multi-agent System

Richard Malek

Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague, Czech Republic
malekr1@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 metaheuristic algorithms simultaneously. By the collaboration of various metaheuristics, we can achieve better results in more classes of problems.
Key words: combinatorial optimization, metaheuristic algorithm, hybrid approach, hyper-heuristic, multi-agent system

1 Introduction

Problems of combinatorial optimization are commonly solved by metaheuristic algorithms.

Combinatorial optimization is the 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.

Best known problems are: Travelling Salesman Problem (TSP), Boolean Satisfiability Problem (SAT) or Job Shop Scheduling Problem (JSSP) [1]. All these problems are NP-complete, thus no effective algorithm exists for them.

Metaheuristic is a high-level strategy for solving optimization problem that guides other heuristics in a search for feasible solutions. Best known metaheuristics are: Genetic Algorithms (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS) and many others [2] [3].

According to the No Free Lunch Theorem the average solution quality provided by any metaheuristic algorithm running on the set of all combinatorial optimization problems is statistically identical [4]. In other words, for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.

Simple implication can be done. To be able to achieve better results in more classes of problems, we should dispose of more algorithms. Recently, this concept is denoted as a hyper-heuristic approach. An objective is that hyper-heuristics will lead to more general systems that are able to handle a wide range of problem domains rather than current metaheuristic technology [5]. We also believe that

the collaborative work of metaheuristic algorithms leads to better results than they could achieve if they worked independently. The solving of the combinatorial optimization problem by various algorithms is outlined in Figure 1.

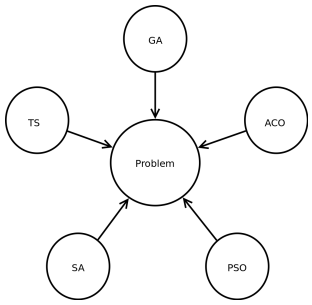


Fig. 1. The solving the combinatorial optimization problem by multiple algorithms

Roughly speaking, a candidate solution is modified by more then one metaheuristic during the search process.

Intuitively, it will ultimately perform at least as well as one algorithm alone, more often perform better, each algorithm providing information to the others to help them. [6]

2 Our Concept of Collaboration

As mentioned above we are interested in using more than one metaheuristic algorithm for solving combinatorial optimization problems.

2.1 Different Metaheuristic, Different Point of View

A different metaheuristic has a different inner representation of the problem it solves and objective function (a function assigning the quality to a candidate solution) as well. Thus, different metaheuristics differ in models of the solution space they seek through and have their own point 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 Metaheuristic Algorithm

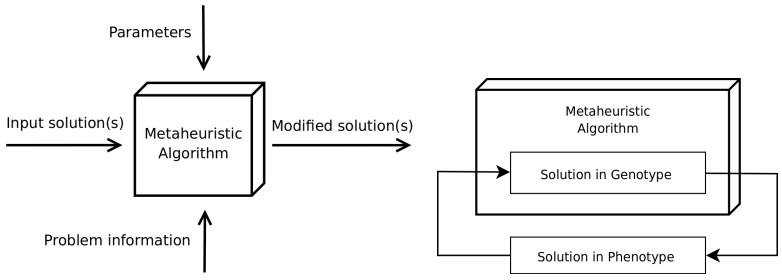
In this section we will describe the interface all metaheuristics can be handled through. This interface unifies an access to all metaheuristics. We apprehend the

metaheuristic 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 metaheuristics, 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 [7][8]. It is a general description of the solution–the form that can be transformed into any other algorithm-specific representations.

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.

From the implementation point of view, the metaheuristic which is added is kept unchanged. Instead of modifying different metaheuristic interfaces into our one (outlined in Figure 2(a)), the only thing one has to do is to implement an adapter for each new metaheuristic. Generally, the adapter is something what converts one interface into another. Then, each metaheuristic is hidden behind the unified adapter interface (see Figure 3).



(a) Algorithm abstraction model (algorithm adapter interface).

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

Fig. 2.

2.3 Population of Candidate Solutions

In our concept, the results of work of particular metaheuristics are shared through a global population of candidate solutions. We denote the place, where the pop-

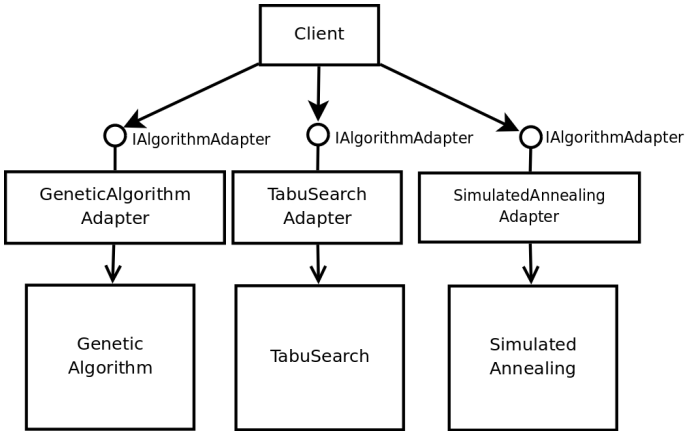


Fig. 3. Handling different metaheuristics (Genetic Algorithm, Tabu Search, Simulated Annealing) through the adapters.

ulation is located, as a solution pool. The candidate solutions are loaded from and stored in the solution pool. Each metaheuristic takes one or more solutions from and after it finishes, puts them back into solution pool. Simply, candidate solutions are improved by many different metaheuristics and population of candidate solutions is being evolved.

3 System Architecture

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 [9] 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 metaheuristics, (global) solution pool and others. Each block is handled by its agent.

Our concept of collaboration of metaheuristic agents is similar to MAGMA [10] approach conceived as a conceptual and practical framework for metaheuristic algorithms.

3.2 Agents

There are following agent types in our system: a problem agent, a solution pool agent, an algorithm agent and an adviser agent.

Figure 4 shows a system configuration with six agents. Arrows represent the agent communication (will be describe in next section).

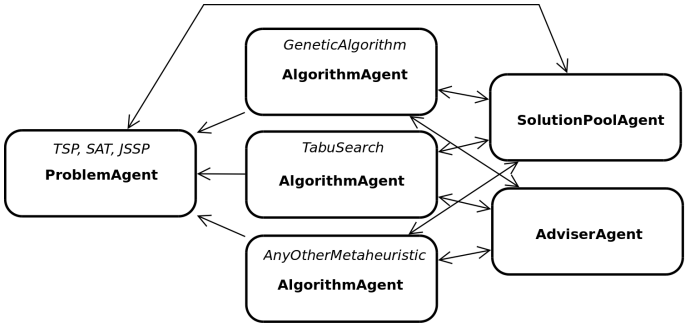


Fig. 4. Agents solving a problem.

Problem Agent - the 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 - the 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 - the agent which disposes of a particular metaheuristic 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 metaheuristic algorithm with received parameters and solution(s)
- sending the best solution found to the problem agent after the metaheuristic algorithm is finished
- sending the solutions back to the solution pool agent
- sending a report on previous algorithm run to the adviser agent

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

Adviser Agent - the agent provides parameter settings for the metaheuristic algorithms and receives reports on algorithm runs. This agent is an object of interest for future work (reasoning).

3.3 System Flow and Agent Messaging

In previous section we described agents and shortly messages they send and receive. Now, we will describe a typical system flow and messages again. Figure 5 illustrates message passing among agents as time flows. There is only one algorithm agent in the figure but the number is not limited.

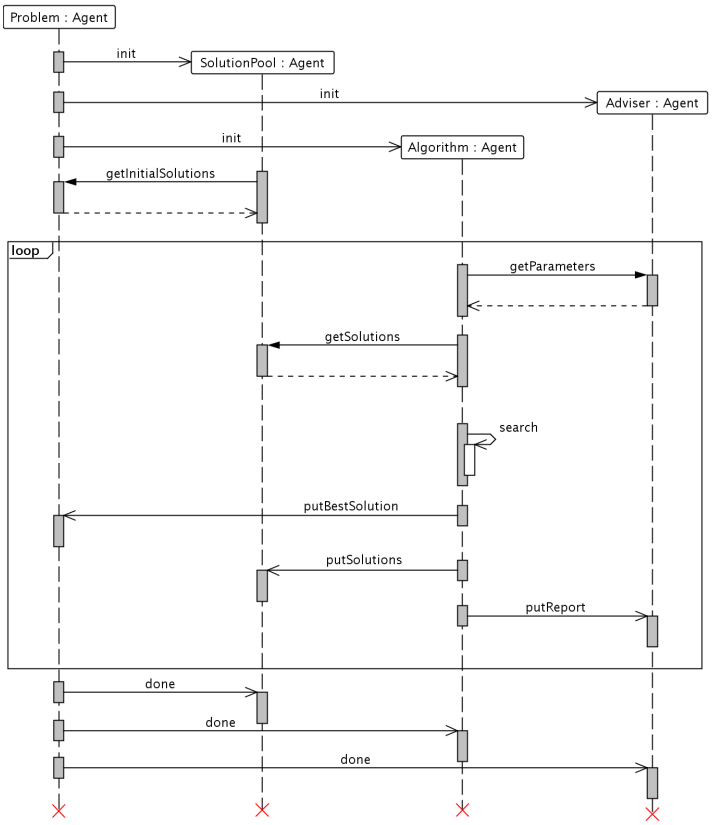


Fig. 5. Sequence diagram of agent message passing

1. The problem agent reads a configuration file with the information about agents it has to create. Default parameter settings for algorithm agents are included in the configuration file as well.

2. The problem agent creates all other agents and prepares (by generating or reading from file) an initial global population.
3. The solution pool agent queries the problem agent and it sends it the initial population back.
4. Once the algorithm agent is created (anytime), it asks the adviser agent for metaheuristic parameters.
5. According to the obtained parameters, the algorithm agent asks the solution pool agent for solutions to work with.
6. The algorithm agent runs its metaheuristic algorithm.
7. After finishing, the algorithm agent sends the best solution to the problem agent and other solutions back to the solution pool agent. The algorithm agent also sends a search report to the adviser agent.
8. The algorithm agent continues by the step 4.
9. After timeout the problem agent sends *done* message to all agents

3.4 System Overview

The overall system architecture can be seen from two points of view.

Component layer point of view (see Figure 6(a)).

1. **Problem** - the package containing all problem-dependent implementations.
2. **Metaheuristics** - the package containing all metaheuristics in their original implementations.
3. **Agents** - the package with agents running within the multi-agent system.
4. **Multi-agent system** A-Globe [9] multi-agent system.

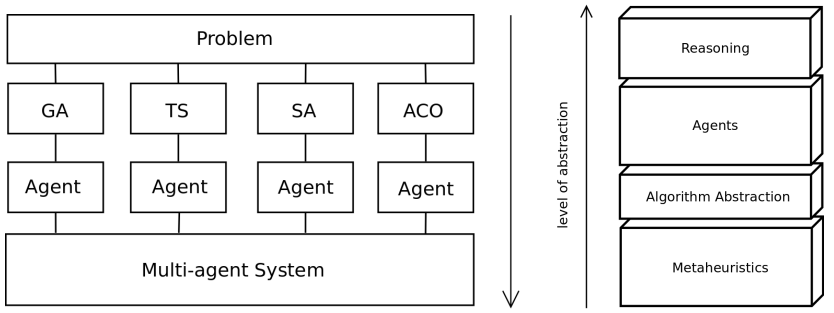
Level of abstraction point of view (see Figure 6(b)).

1. **Reasoning Layer** - the package containing a parameter reasoning implementation (see section 5)
2. **Agent Layer** - the package contains implementation of all agents mentioned above and their communication model as well.
3. **Algorithm Abstraction Layer** - the package Layer contains implementation of adapters of all metaheuristics we have at disposal.
4. **Metaheuristic Layer** - the package containing all metaheuristics in their original implementations.

From the problem perspective, each layer represents a different level of abstraction. The higher level means more general tasks.

3.5 Main Advantages of the Approach Based on Multi-agent System

Extensibility System architecture based on multi-agent system paradigm simplify extensibility of the whole system. No inner complex bindings among components, just agents and the communication model. Adding another system feature is just about the implementation of a new agent.



(a) Component layer point of view

(b) Level of abstraction point of view

Fig. 6. System overview.

Scalability is a feature of MAS allowing us freely to add or remove agents even during runtime. There are no limits (except HW limitations) for the number of algorithm agents we can run. Practically, one algorithm per one processor seems to be reasonable. Otherwise distribution of MAS comes into account.

Distributability Multi-agent systems typically contain agent containers–platforms which can be distributed on remote machines and utilize host resources. Communication among agents from different platforms is the same as within one platform. This allows the creation of large logical multi-agent system running on many computers.

4 Experiments

For experiment purposes, we have implemented Genetic Algorithm (GA) and Tabu Search (TS) metaheuristics and tested them on Traveling Salesman Problem (TSP). We run GA and TS on several TSP benchmark instances with parameters experimentally set in previous experiments (not mentioned here). Each algorithm was run 10x and the best and the average values of found solutions have been recorded. Then we run TS and GA simultaneously. In all three cases, the algorithms were run for the same time–10 minutes. The results are presented in Figure 4.

We have got the best results by algorithm collaboration in most cases and we found the results promising for future work.

5 Future Work

5.1 Reasoning

Adviser Agent The effectiveness of a metaheuristic algorithm solving a given problem depends on parameters that were set. Values of the parameters are

		GA		TS		GA+TS	
instance	optimal	min	avg	min	avg	min	avg
eil51	426	478	512	430	438	**428**	**435**
berlin52	7542	8261	9329	8007	8139	**7544**	**7693**
ch130	6110	11300	12074	9315	9644	**7303**	**7829**
ch150	6528	13564	14837	9754	10224	**8118**	**8654**
d198	15780	38177	44957	25393	28771	**24235**	**26074**
gil262	2378	9374	9670	4434	4765	4 **223**	**4462**
lin318	42029	216098	228465	99373	103442	**90673**	**100091**
rd400	15281	92349	97823	**39258**	44567	40682	**44239**
pr439	107217	791144	846171	**345676**	418296	386018	**413126**
u574	36905	345138	352248	**156576**	**207538**	214900	216475

Fig. 7. Table of results. Best values are printed bold.

commonly determined experimentally and they are not changed during the algorithm runtime. The problem is that "optimal" parameters may differ with different data sets. Instead of determining the parameters by hands, it would be nice to set the parameter values automatically. In other words, we are interested in the evolution of parameters. That's what the adviser agent should do (see section 3.2 for information about the role of the adviser agent).

We remark that algorithm agents ask the adviser agents for metaheuristic parameters and send back reports describing metaheuristic algorithm runs. According to the reports, the adviser answers for the parameter requests. It may answer randomly or it can create its inner model supporting the parameter reasoning.

During the algorithm runtime a lot of information may be tracked. For example: the number of solution improvements, the number of iterations without change, the change of average solution quality, etc.All the information may indicate if the algorithm parameters were set properly. The idea is such that we will track all the runtime information and when the algorithm stops, the information will be processed and new parameter values will be set accordingly.

Metaheuristic Competition Because of limited resources and possibly unlimited number of algorithm agents, we have to select which combination of metaheuristics and in how many instances (algorithm agents) is employed for solving a given optimization problem. In this context a competition of algorithm agents is intended.

6 Conclusion

In this paper we have presented our concept and the implementation solving the problems of combinatorial optimization by employing various metaheuristic algorithms. Being aware of No Free Lunch theorem, our concept is based on

combination of particular metaheuristics to achieve better results for wide scale of problems and problem instances as well.

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

The metaheuristic algorithm is apprehended 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. On input and output it handles with candidate solutions in phenotype (algorithm independent) representation. This abstraction allows algorithm chaining when the output of one algorithm can be put on the input of the other one.

For this purpose, an architecture based on multi-agent system paradigm has been chosen. Algorithm agents and the solution pool agent are the base of the system. Main advantages of the multi-agent system are: scalability, extensibility and distibutability.

Two algorithms have been implemented (GA, TS) and tested on Travelling Salesman Problem. Experiments have shown that improvements of solution quality obtained by the metaheuristic collaboration are notable. However, other detailed experiments have to be done.

Automated parameter setting of particular algorithms was discussed and the adviser agent solving the parameter adaptation was proposed. It is also the object of interest for future work.

References

1. Skiena, S.S.: The Algorithm Design Manual. Springer-Verlag (1998)
2. Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Frome, BA11 6TT, United Kingdom (2008)
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. Talbi, E.G.: A taxonomy of hybrid metaheuristics. Journal of Heuristics **8**(5) (2002) 541–564
7. Fogel, D.B.: Phenotypes, genotypes, and operators in evolutionary computation. (1995) 193–198
8. Back, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK, UK (1997)
9. 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
10. Roli, A., Milano, M.: Magma: A multiagent architecture for metaheuristics. IEEE Trans. on Systems, Man and Cybernetics - Part B **34** (2002) 2004