Global Optimization through Meta-Heuristic Collaboration in a Multi-Agent System

Richard Málek

August 28, 2009

Contents

1 Introduction **3**
1.1 Concept of Our Approach . . . 3
1.2 Document Structure . . . 4

2 Background **5**
2.1 Optimization . . . 5
2.1.1 Definitions . . . 5
2.1.1.1 Optimization Problem . . . 5
2.1.1.2 Single Objective Functions . . . 6
2.1.1.3 Multiple Objective Functions . . . 6
2.1.2 Optimization Classification . . . 7
2.2 Review of the State of the Art . . . 10
2.2.1 Heuristics . . . 10
2.2.2 Meta-Heuristics . . . 10
2.2.3 Meta-Evolutionary Approaches . . . 10
2.2.4 Taxonomy of Hybrid Approaches . . . 10
2.2.5 From Meta-Heuristics to Hyper-Heuristics . . . 11
2.2.6 Hyper-Heuristics . . . 11

3 Proposed Approach **13**
3.1 Motivation . . . 13
3.2 Concept of Meta-Heuristic Collaboration . . . 13
3.2.1 Different Meta-Heuristic, Different Point of View . . . 13
3.2.2 Abstraction of Meta-Heuristic Algorithm . . . 13
3.2.3 Population of Candidate Solutions . . . 15
3.3 System Architecture . . . 15
3.3.1 The Use of Multi-Agent System . . . 15
3.3.2 Agents . . . 15
3.3.2.1 Problem Agent . . . 15
3.3.2.2 SolutionPool Agent . . . 16
3.3.2.3 Algorithm Agent . . . 16
3.3.2.4 Adviser Agent . . . 16
3.3.3 System Flow and Agent Messaging . . . 16
3.3.4 System Overview . . . 17
3.3.4.1 Component layer . . . 17
3.3.4.2 Level of abstraction . . . 18
3.3.5 Main Advantages of the Approach Based on Multi-agent System . . . 18
3.4 Experiments . . . 19

4 Related Work **20**
4.1 Combination of Base-Level Meta-Heuristics 20
4.2 Multi-Agent Approach to Base-Level Meta-Heuristic Combination 20
4.3 Pure Hyper-Heuristic Approach 20
4.4 Multi-Agent Hyper-Heuristic Approach 20
4.5 Comparison to Our Approach 21

5 Conclusions and Future Work **22**
5.1 Conclusions 22
5.1.1 Current Work Accomplishments 22
5.2 Future Work 22
5.2.1 Works under the Problem Barrier 22
5.2.2 Works above the Problem Barrier 23
5.2.2.1 Parameter Setting Reasoning 23

Chapter 1

Introduction

Optimization [1][2][3] is a process of finding function extrema to solve problems in which the quality of any answer can be expressed as a numerical value. Such problems arise in all areas of mathematics, the physical, chemical and biological sciences, engineering, architecture, economics, and management, and the range of techniques available to solve them is nearly as wide. Obviously, providing solutions as good as possible, would have a non-negligible economical effect in all the areas mentioned above.

In *global optimization* [4], the goal is to find a solution which is the best of all defined on the whole problem space (a set of all possible combinations of values). The complexity of the optimization problems differs significantly and actually it is given by the complexity of problem space (or state, search space) structure. For some problems, whose problem space structure can be described analytically or are constrained enough, efficient algorithms exist. The algorithms usually work deterministically and provide solution in reasonable time even for large problem instances.

Unfortunately, most practical optimization problems are too complex 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.

In our research, combining methods of non-deterministic, probabilistic optimization is exactly the object of our interest.

1.1 Concept of Our Approach

The basic idea of our approach is to solve an optimization problem by various meta-heuristic algorithms simultaneously - as depicted in Figure 1.1.

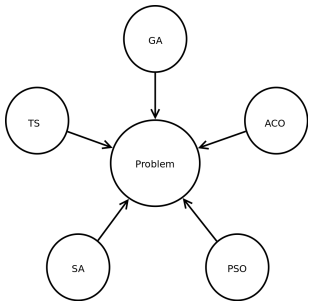


Figure 1.1: The solving an optimization problem by various meta-heuristic algorithms.

Each meta-heuristic modifies one or many candidate solutions and shares them with other meta-heuristics through a global pool of candidate solutions. For this purposes, a parallel envi-

ronment with message passing is essential. The multi-agent system provides us such an environment. Each meta-heuristic and the solution pool are represented by their own agents which can work simultaneously and send messages to each other.

Recent developments in search methodologies towards more generally applicable techniques has been termed *hyper-heuristics* [5]. 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. Such a characteristic is suitable for our approach as well.

Briefly speaking, our *hyper-heuristic* approach 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 upon the multi-agent system.

1.2 Document Structure

The first chapter introduces this work.

The second chapter provides background information on optimization problem, optimization classification and optimization techniques.

The third chapter describes our approach in details.

The fourth chapter presents the related work and compares it to our approach.

The fifth chapter concludes current work and summarizes the future work.

Chapter 2

Background

2.1 Optimization

Optimization problems are made up of three basic ingredients:

- An *objective function* which we want to *minimize* or *maximize*.
- A set of *unknowns* or *variables* which affect the value of the objective function.
- A set of *constraints* that allow the unknowns to take on certain values but exclude others.

Informally, the optimization problem is: Find values of the *variables* that *minimize* or *maximize* the *objective function* while satisfying the *constraints* [6].

2.1.1 Definitions

2.1.1.1 Optimization Problem

The goal of an optimization problem can be formulated as follows: find the combination of parameters (independent variables) which optimize a given quantity, possibly subject to some restrictions on the allowed parameter ranges. The quantity to be optimized (maximized or minimized) is termed the *objective function*; the parameters which may be changed in the quest for the optimum are called *control* or *decision variables*; the restrictions on allowed parameter values are known as *constraints*. The general optimization problem may be stated mathematically as:

Definition 2.1. (Optimization Problem).

$$\begin{array}{lllll} \text{given a function} & f : \mathbf{A} \mapsto \mathbb{R} & & & \mathbf{A} \subseteq \mathbb{R}^n \\ \text{minimize} & f(\mathbf{x}), & \mathbf{x} = (x_1, x_2, ..., x_n)^T & & \mathbf{x} \in \mathbf{A} \\ \text{subject to} & c_i(\mathbf{x}) = 0, & i = 1, 2, ..., m' & & i, m' \in \mathbb{N} \\ & c_i(\mathbf{x}) \geq 0, & i = m' + 1, ..., m & & i, m \in \mathbb{N} \end{array}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x}$ is the column vector of the n independent variables, and $c_i(\mathbf{x})$ is the set of constraint functions. Constraint equations of the form $c_i(\mathbf{x}) = 0$ are termed *equality* constraints, and those of the form $c_i(\mathbf{x}) \geq 0$ are *inequality* constraints. Taken together, $f(\mathbf{x})$ and $c_i(\mathbf{x})$ are known as the *problem functions* [7].

Typically, $\mathbf{A}$ is some subset of the Euclidean space $\mathbb{R}^n$, specified by a set of constraints, equalities or inequalities that the members of $\mathbf{A}$ have to satisfy. The domain $\mathbf{A}$ of f is called the search space or the choice set, while the elements of $\mathbf{A}$ are called candidate solutions or feasible solutions.

2.1.1.2 Single Objective Functions

Definition 2.2. (Local Maximum). A (local) maximum $\check{x}_l \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{x}_l) \geq f(x)$ for all x neighbouring $\check{x}_l$.

If $\mathbb{X} \subseteq \mathbb{R}^n$, we can write:

$$\forall \check{\mathbf{x}}_l \exists \varepsilon > 0 : f(\check{\mathbf{x}}_l) \geq f(\mathbf{x}) \; \forall \mathbf{x} \in \mathbb{X}, |\mathbf{x} - \check{\mathbf{x}}_l| < \varepsilon$$

Definition 2.3. (Local Minimum). A (local) minimum $\hat{x}_l \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{x}_l) \leq f(x)$ for all x neighboring $\hat{x}_l$.

If $\mathbb{X} \subseteq \mathbb{R}^n$, we can write:

$$\forall \hat{\mathbf{x}}_l \exists \varepsilon > 0 : f(\hat{\mathbf{x}}_l) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in \mathbb{X}, |\mathbf{x} - \hat{\mathbf{x}}_l| < \varepsilon$$

Definition 2.4. (Local Optimum). A (local) optimum $x_l^\star \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is either a local maximum or a local minimum.

Definition 2.5. (Global Maximum). A global maximum $\hat{\mathbf{x}} \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{\mathbf{x}}) \geq f(\mathbf{x}) \; \forall \mathbf{x} \in \mathbb{X}$.

Definition 2.6. (Global Minimum). A global maximum $\check{\mathbf{x}} \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{\mathbf{x}}) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in \mathbb{X}$.

Definition 2.7. (Global Optimum). A global optimum $x^\star \in \mathbb{X}$ of an (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is either a global maximum or a global minimum.

Definition 2.8. (Global Optimization). A process of finding of a global optimum.

2.1.1.3 Multiple Objective Functions

In many real-world applications, optimization algorithms are applied to sets F consisting of $n = |F|$ objective functions f_i, each representing one criterion to be optimized. Algorithms designed to optimize such sets of objective functions are usually named with the prefix *multi-objective*.

Formally:

$$F = \{f_i : \mathbb{X} \mapsto Y_i : 0 < i \leq n, Y_i \subseteq \mathbb{R}\}$$

Weighted Sum (Linear Aggregation)
The simplest method to define what is optimal is computing a weighted sum $g(x)$ of all the functions $f_i(x) \in F$ according to the following formula:

$$g(x) = \sum_{\forall f_i \in F} w_i f_i(x)$$
$$\mathbf{x}^\star \in \mathbb{X}^\star \Leftrightarrow g(\mathbf{x}^\star) \geq g(\mathbf{x}) \; \forall \mathbf{x} \in \mathbb{X}$$

Pareto Optimization

Definition 2.9. (Domination). An element x_1 dominates (is preferred to) an element x_2 ($x_1 \vdash x_2$) if x_1 is better than x_2 in at least one objective function and not worse with respect to all other objectives. Based on the set F of objective functions f, we can write:

$$x_1 \vdash x_2$$
$$\overset{\Leftrightarrow}{}$$
$$\forall i : 0 < i \leq n \Rightarrow \omega_i f_i(x_1) \leq \omega_i f_i(x_2) \wedge$$
$$\exists j : 0 < j \leq n \Rightarrow \omega_j f_j(x_1) < \omega_j f_j(x_2)$$
$$\omega_i = \begin{cases} 1 \; if \; f_i \; should \; be \; minimized \\ -1 \; if \; f_i \; should \; be \; maximized \end{cases}$$

Definition 2.10. (Pareto Optimal). An element $\mathbf{x}^\star \in \mathbb{X}$ is Pareto optimal (and hence, part of the optimal set $\mathbb{X}^\star$) if it is not dominated by any other element in the problem space $\mathbb{X}$. In terms of Pareto optimization, $\mathbb{X}^\star$ is called the Pareto set or the Pareto Frontier.

$$\mathbf{x}^\star \in \mathbb{X}^\star \Leftrightarrow \neg\exists \mathbf{x} \in \mathbb{X} : \mathbf{x} \vdash \mathbf{x}^\star$$

2.1.2 Optimization Classification

There are so many different kinds of optimization tasks, and thus so many different methods solving the tasks. The methods can be classified from the following aspects.

Global vs. Local Optimization
Global optimization is the task of finding the absolutely best set of admissible conditions to achieve the objective, formulated in mathematical terms. At present, no efficient deterministic algorithm is known for performing this task in general. For many applications, local minima are good enough.

Intuitively, *local optimization* is the task of finding the best solution in a subset of all possible solutions.

Discrete vs. Continuous Optimization
The generic term *discrete* optimization usually refers to problems in which the solution we seek is one of a number of objects in a finite set and the variables make sense only if they take on integer values.

By contrast, *continuous* optimization problems find a solution from an uncountably infinite set– typically a set of vectors with real components.

Derivative vs. Non-Derivative Optimization
Direct techniques are based on the *derivative* method in which the gradient of an objective function is calculated and set to zero to obtain points at which its value becomes maximum or minimum. The second derivative of the function is then used to determine whether the points so obtained are maxima or minima. Clearly, therefore, the objective function must be continuous and twice differentiable. This requirement is generally not met in real world problems. Classical techniques however, have the advantage of being mathematically neat, simple to understand and easy to use [2].

Non-derivative optimization problems are defined by objective functions for which derivatives are unavailable or available at a prohibitive cost. Thus, such methods only use values of objective function without any derivation information. Derivative-free optimization is currently an area of great demand - as well as in this work.

Constrained vs. Unconstrained Optimization
Constrained optimization problems arise from models that include explicit constraints on the variables. These constraints may be simple bounds such as $0 \leq x_1 \leq 100$, more general linear constraints such as $\sum_i x_i \leq 1$, or non-linear inequalities that represent complex relationships among the variables.

In *unconstrained* optimization, an objective function depends on real variables, with no restrictions at all on the values of these variables. Unconstrained problems arise also as reformulations of constrained optimization problems, in which the constraints are replaced by penalization terms in the objective function that have the effect of discouraging constraint violations.

Deterministic vs. Non-Deterministic (Stochastic, Probabilistic) Optimization
In *deterministic* optimization, the model[1] is fully specified. Deterministic also means that methods do not contain any random elements. The optimization sub-field dealing with deterministic methods is called Numerical Optimization [1].

Non-deterministic optimization comes into play when the model cannot be fully specified because it depends on quantities that are unknown at the time of formulation[2] or the dimensionality of the search space is very high. Then, trying to solve a problem deterministically would possibly result in an exhaustive enumeration of the search space, which is not feasible even for relatively small problems.

The major optimization sub-fields (partially compliant with the previous aspects) are: Linear programming, Integer programming, Quadratic programming, Non-linear programming, Convex programming, Semi-definite programming, Stochastic programming, Combinatorial optimization, Constraint programming, Dynamic programming. The optimization classification is summarized in Table 2.1.

Characteristic	**Property**	**Classification**
Number of control variables	One	Univariate
	More than one	Multivariate
Type of control variables	Continuous real numbers	Continuous
	Integers	Integer or Discrete
	Both continuous real numbers and integers	Mixed Integer
	Integers in permutations	Combinatorial
Problem functions	Linear functions of the control variables	Linear
	Quadratic functions of the control variables	Quadratic
	Other nonlinear functions of the control variables	Non-linear
Problem formulation	Subject to constraints	Constrained
	Not subject to constraints	Unconstrained

Table 2.1: Optimization classification

In this work, we are focused mainly on methods of global, probabilistic (non-deterministic, non-derivative) optimization–as depicted in Figure 2.1.

[1] A model is an abstraction or approximation of a system that allows to reason and to deduce properties of the system.

[2] The objective function is considered as a black box function, i.e. a function for which we do not know an algebraic expression.

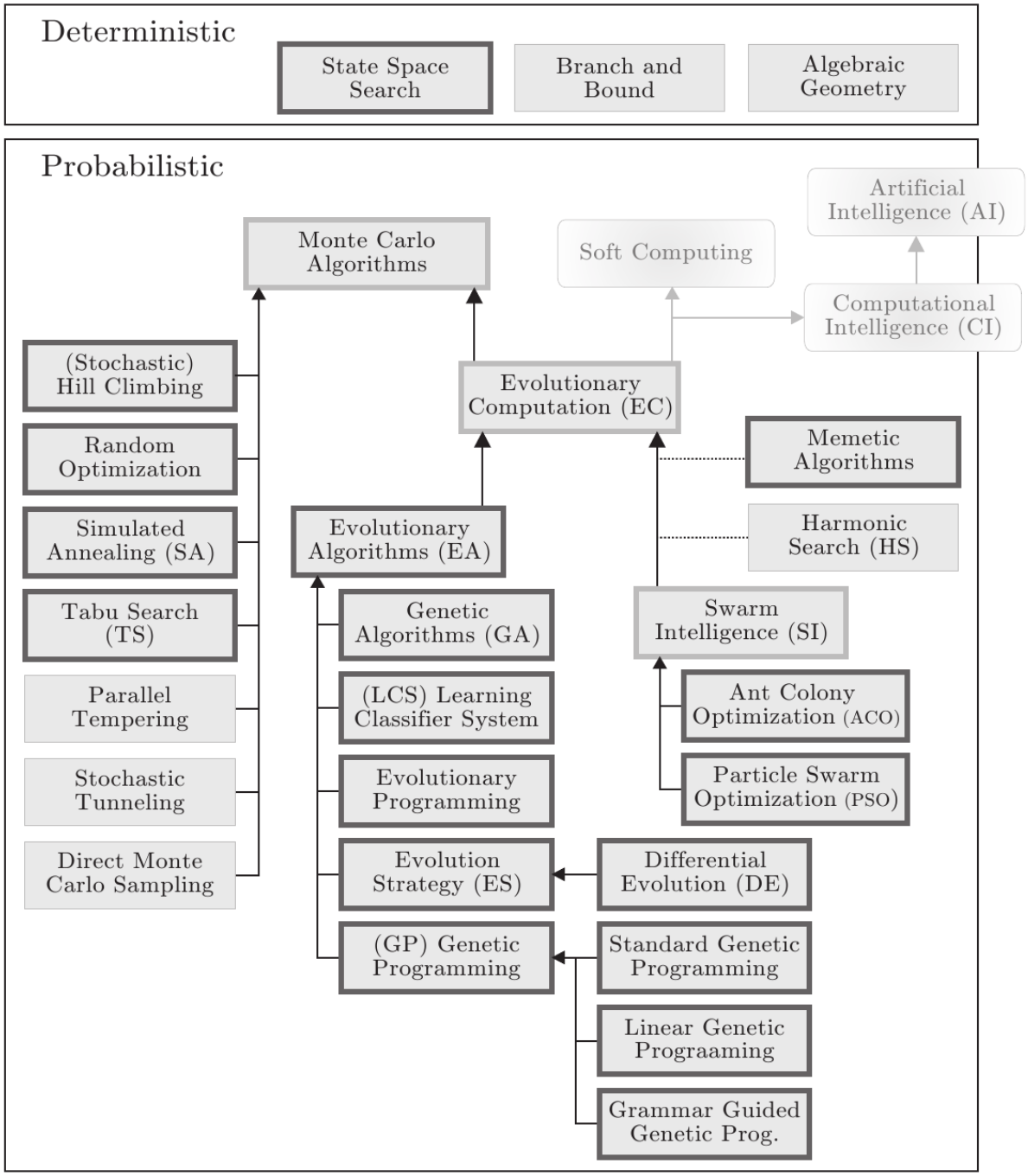


Figure 2.1: Optimization classification - a taxonomy of probabilistic methods [4].

2.2 Review of the State of the Art

2.2.1 Heuristics

The term *heuristic* [8] is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found. A *heuristic* is a part of an optimization algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced. Heuristics are usually problem class dependent.

2.2.2 Meta-Heuristics

A *meta-heuristic* [9][10] is a high-level strategy for solving optimization problem that guides other heuristics in a search for feasible solutions.

Meta-heuristics solve very general classes of problems. For many practical problems, meta-heuristic algorithms may be the only way to get good solutions in a reasonable amount of time.

Meta-heuristics such as Genetic Algorithms (GA), Simulated Annealing (SA), Evolutionary Algorithms (EA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Tabu Search (TS) [11][12] [13] have been successfully applied to many difficult optimization problems for which no satisfactory problem specific solution exists.

There is no straightforward answer to the question which algorithm should be used for given problem and how to configure it. Therefore, expertise is required to adopt a meta-heuristic for solving a problem in a certain domain.

2.2.3 Meta-Evolutionary Approaches

The basic idea of meta-evolutionary approach [14] is to consider the search for the best evolutionary algorithm [15] as an optimization problem and use another evolutionary algorithm to solve it. In other words, to determine the best evolutionary algorithms, that means to determine the best evolutionary operators and their parameters settings such as: population size, the crossover probability, and the mutation probability, another evolutionary algorithm is used. Typical representatives can be found in [16, 17, 18]. The base-level EA parameters can be either *tuned* or *controlled* [19]. The parameter tuning fixes the parameters for the whole base-level EA run. On the other hand the parameter control continually adjusts parameters throughout the evolution. The controlled parameters can be either *adapted* using a predetermined rule or a *self-adaptation* can be employed [20]. Here, the parameters to be adapted are encoded into a chromosome of an individual and undergo a mutation and a recombination. Other good source of information concerning adaptation can be [21].

Memetic algorithms (MA) [22] are evolutionary based algorithms that apply a local search process to refine solutions of hard problems [23] – MAs are a particular class of global-local search hybrids. Traditionally, MAs with Lamarckian learning procedures are based on the use of a evolutionary algorithm and a single local search method for local improvements [24]. *Meta-Lamarckian* learning in MAs uses a set of several local search methods, choosing the proper method for given problem to achieve an improved search performance.

2.2.4 Taxonomy of Hybrid Approaches

The performance of one method is often successfully improved by using principle of another method. The methods grow through each other and the hybrid [25] of two (or more) methods then arise. According to the taxonomy described in [26] [27] the process of growing through is denoted as a low-level hybridization. A high-level hybridization is in opposite to this. It denotes

the use of various meta-heuristic algorithms on searching the solution cooperatively. See Figure 2.2.

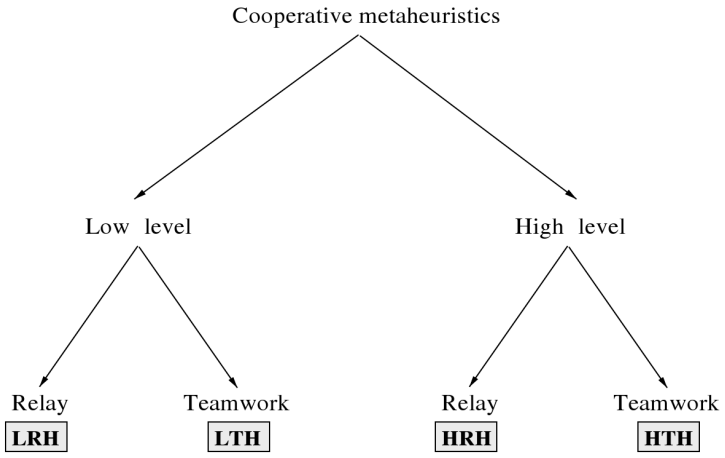


Figure 2.2: Taxonomy of cooperative meta-heuristics

2.2.5 From Meta-Heuristics to Hyper-Heuristics

Meta-heuristics have played a key role at the area of global optimization over the last 10-15 years or so. The term 'meta-heuristic' is sometimes used to refer to a whole search algorithm. Viewing meta-heuristics as search algorithms, some authors have occasionally tried to argue for the absolute superiority of one heuristic over another. This practice started to die out when in 1995 Wolpert and MacReady [28] published their No Free Lunch Theorem which showed that, when averaged over all problems defined on a given finite search space, all search algorithms had the same average performance. The No Free Lunch Theorem helped to focus attention on the question of what sorts of problems any given algorithm might be particularly useful for [5]. This also led to the concept of hyper-heuristics.

2.2.6 Hyper-Heuristics

Hyper-heuristics [5][29][30][31] introduce a novel approach for search and optimization.

The term hyper-heuristic was first introduced by Cowling et al. in [32]. They defined a hyper-heuristic as an approach that operates at a higher level of abstraction than meta-heuristics and manages the choice of which low-level heuristic method should be applied at any given time, depending upon the characteristics of the region of the solution space currently under exploration. This means that the hyper-heuristic itself does not search for a better solution to the problem. Instead, it selects at each step of the solution process the most promising simple low-level heuristic (or combination of heuristics) which is potentially able to improve the solution.

Low-level heuristics usually represent the simple local search neighbourhoods or the rules used by human experts for constructing solutions. However, it is also possible that more complex heuristics such as meta-heuristics can be considered at a lower level. All domain-specific information is concentrated in the set of low-level heuristics and the objective function. Hyper-heuristics do not require knowledge of how each low-level heuristic works or the contents of the objective function of the problem (other than the value returned) [30].

The Figure 2.3 shows that there is a barrier between the low level heuristics and the hyper-heuristic. Domain knowledge is not allowed to cross this barrier. Therefore, the hyper-heuristic

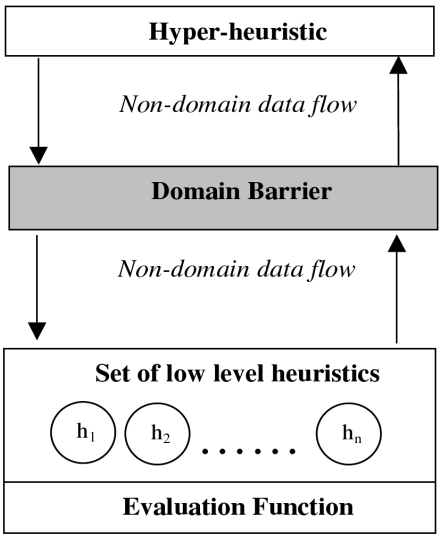


Figure 2.3: Hyperheuristic framework

has no knowledge of the domain under which it is operating [5].

It is clear that hyper-heuristic development is going to play a major role in search technology over the next few years. The potential for scientific progress in the development of more general optimisation systems, for a wide variety of application areas, is significant [5].

Chapter 3

Proposed Approach

3.1 Motivation

According to the No Free Lunch Theorem the average solution quality provided by any meta-heuristic algorithm running on the set of all combinatorial optimization problems is statistically identical [28]. 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 meta-heuristic technology [5]. We also believe that the collaborative work of meta-heuristic 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 1.1.

Roughly speaking, a candidate solution is modified by more then one meta-heuristic 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. [26]

3.2 Concept of Meta-Heuristic Collaboration

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

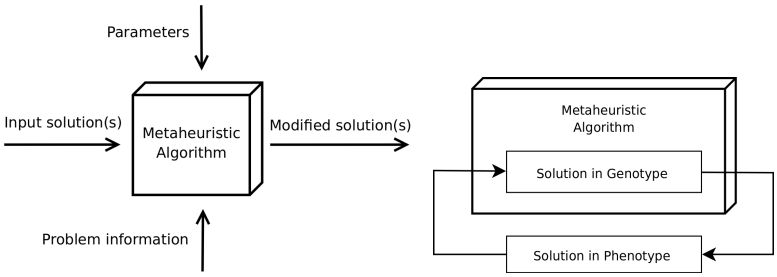
3.2.1 Different Meta-Heuristic, Different Point of View

A different meta-heuristic 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 meta-heuristics 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.

3.2.2 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



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

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

Figure 3.1:

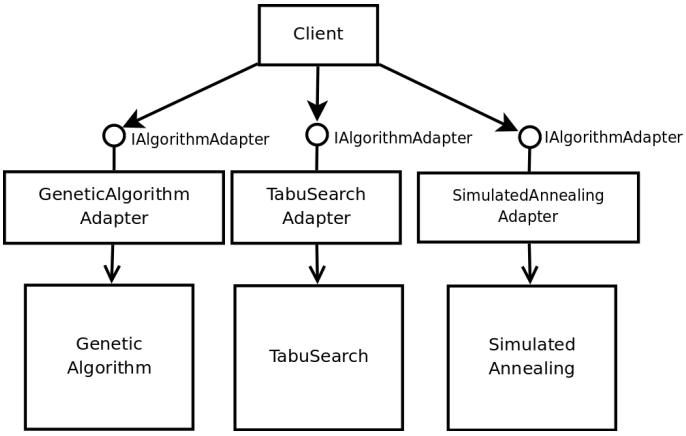


Figure 3.2: Handling different meta-heuristics (Genetic Algorithm, Tabu Search, Simulated Annealing) through the adapters.

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 3.1(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 [33][15]. 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 3.1(b). This evolves chaining of any algorithms that can do the phenotype/genotype mapping.

From the implementation point of view, the meta-heuristic which is added is kept unchanged. Instead of modifying different meta-heuristic interfaces into our one (outlined in Figure 3.1(a)), the only thing one has to do is to implement an adapter for each new meta-heuristic. Generally, the adapter is something what converts one interface into another. Then, each meta-heuristic is hidden behind the unified adapter interface (see Figure 3.2).

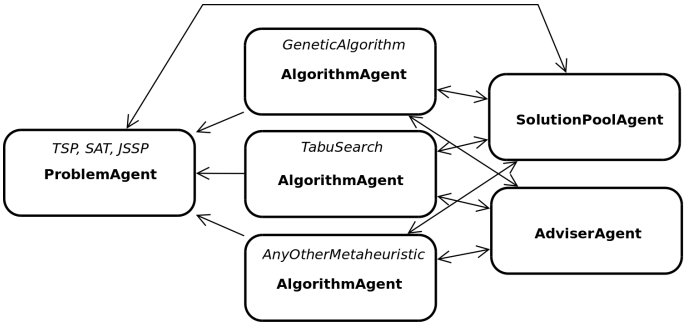


Figure 3.3: Agents solving a problem.

3.2.3 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.

3.3 System Architecture

3.3.1 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 [34] 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.

Our concept of collaboration of meta-heuristic agents is similar to MAGMA [35] approach conceived as a conceptual and practical framework for meta-heuristic algorithms.

3.3.2 Agents

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

Figure 3.3 shows a system configuration with six agents. Arrows represent the agent communication (will be describe in next section).

3.3.2.1 Problem 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.

3.3.2.2 SolutionPool Agent

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.

3.3.2.3 Algorithm Agent

Algorithm agent which 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 run to the adviser agent

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

3.3.2.4 Adviser Agent

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

3.3.3 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 3.4 illustrates message passing among agents as time flows. There is only one algorithm agent in the figure but the number is not limited.

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 meta-heuristic parameters.
5. According to the obtained parameters, the algorithm agent asks the solution pool agent for solutions to work with.

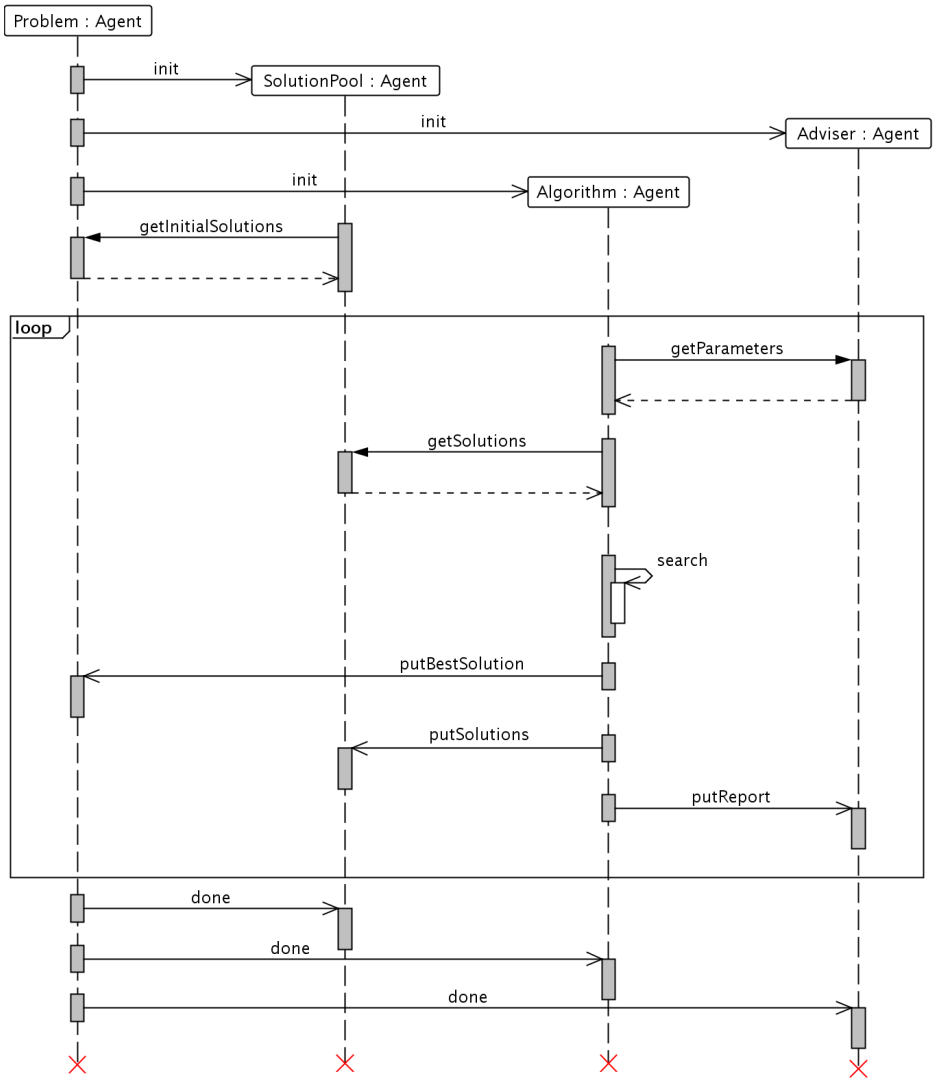


Figure 3.4: Sequence diagram of agent message passing

6. The algorithm agent runs its meta-heuristic 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

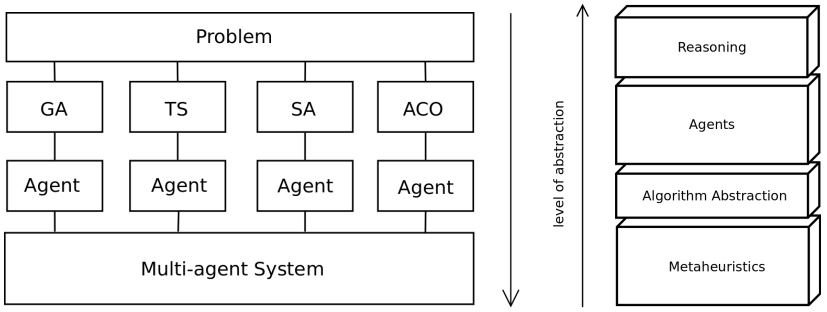
3.3.4 System Overview

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

3.3.4.1 Component layer

point of view (see Figure 3.5(a)).

1. **Problem** - the package containing all problem-dependent implementations.



(a) Component layer point of view

(b) Level of abstraction point of view

Figure 3.5: System overview.

2. **Meta-heuristics** - the package containing all meta-heuristics in their original implementations.
3. **Agents** - the package with agents running within the multi-agent system.
4. **Multi-agent system** A-Globe [34] multi-agent system.

3.3.4.2 Level of abstraction

point of view (see Figure 3.5(b)).

1. **Reasoning Layer** - the package containing a parameter reasoning implementation (see section 5.2)
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 meta-heuristics we have at disposal.
4. **Meta-heuristic Layer** - the package containing all meta-heuristics in their original implementations.

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

3.3.5 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.

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.

3.4 Experiments

For experiment purposes, we have implemented Genetic Algorithm (GA) and Tabu Search (TS) meta-heuristics and tested them on Travelling 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 Table 3.1.

		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

Table 3.1: Table of results. Best values are printed bold.

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

Chapter 4

Related Work

In this chapter, selected publications similar to our approach are presented. Names of following paragraphs are chosen according to similarity aspects.

4.1 Combination of Base-Level Meta-Heuristics

In [36], authors switch three optimization algorithms: DIRECT (DIviding RECTangles), GA (Genetic Algorithm) and SQP (Sequential Quadratic Programming). The algorithms are run sequentially starting with the DIRECT global search method, then improving accuracy and narrowing down the search area by GA and finally finishing with fine-tuning SQP. The switch to a more local-oriented search is realized when a predefined termination criterion is met. These criteria involve: limits on a number of iterations, diversity measure or getting close enough to the known value of the optimum.

In our previous work [37], we have developed a similar strategy for discrete problems and the results of switching genetic algorithm with tabu search are promising.

4.2 Multi-Agent Approach to Base-Level Meta-Heuristic Combination

MAGMA [38][35] 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.

4.3 Pure Hyper-Heuristic Approach

In [39], frameworks based on the general hyper-heuristic approach have been defined in order to make better use of hill climbers as heuristics. In the study, four different frameworks are used; F_A, F_B, F_C and F_D, as summarized in Figure 4.1 .

4.4 Multi-Agent Hyper-Heuristic Approach

In [40], 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

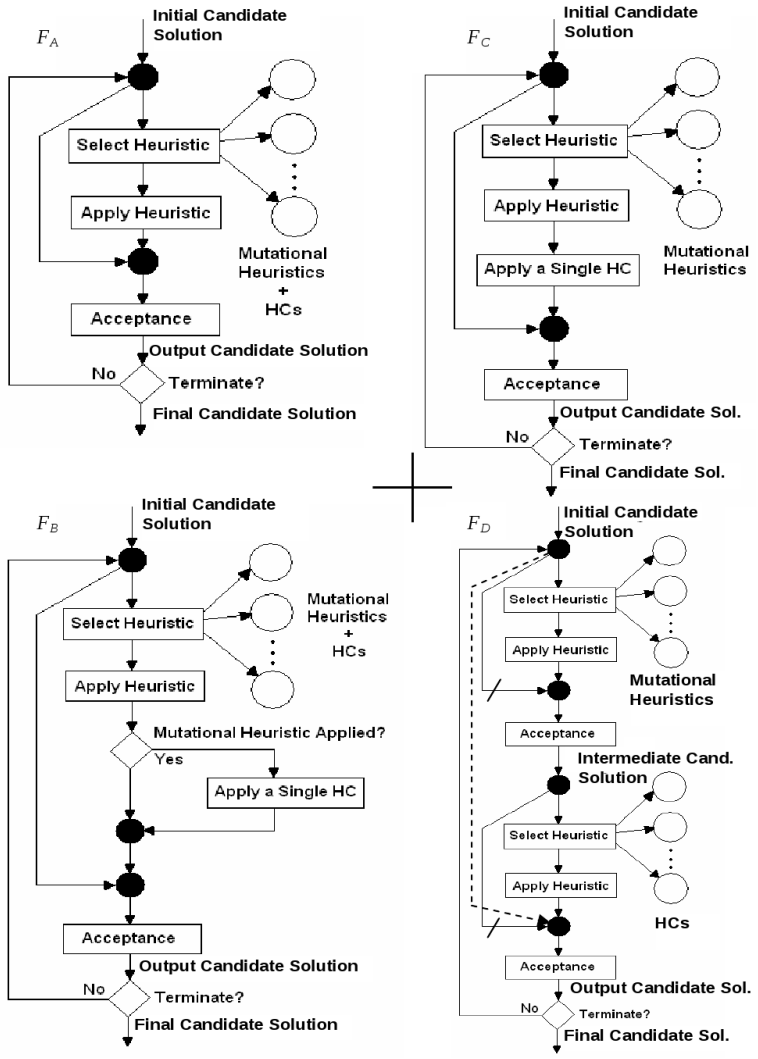


Figure 4.1: Different hyperheuristic frameworks combining mutational heuristics and hill climbers.

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.

4.5 Comparison to Our Approach

Our approach is considerably similar to a concept of hyper-heuristic framework. In our case, a problem barrier (as depicted in Figure 2.3) is represented by Algorithm Abstraction Layer described in section 3.3.4.2. In our concept, however, there are two main differences: First, most of current hyper-heuristic frameworks use relatively low-level (problem specific) heuristics, whereas we use meta-heuristics. Second, hyper-heuristic algorithms create sequences of heuristic applications, whereas our meta-heuristics work simultaneously.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this work, we have presented our concept of meta-heuristic algorithm collaboration. The concept is considerably similar to a hyper-heuristic approach, but not entirely. Differences have been discussed in section 4.5.

The combination of particular meta-heuristics is intended to achieve better results for wide scale of optimization problems and problem instances as well. To be such a implementation possible to realize, an abstraction of the meta-heuristic algorithm had to be done.

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. 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.

5.1.1 Current Work Accomplishments

We have implemented a basic functionality of proposed approach upon the Aglobe multi-agent system. Now, we are able to run (possibly unlimited) numbers of algorithm agents holding different meta-heuristics. Two meta-heuristic algorithms have been implemented (GA, TS) and tested on Travelling Salesman Problem. Experiments have shown that improvements of solution quality obtained by the meta-heuristic collaboration are notable. However, other detailed experiments have to be done.

5.2 Future Work

5.2.1 Works under the Problem Barrier

The problem barrier term is meant consistently with the term of hyper-heuristic approach (see Figure 2.3).

Implementation of other meta-heuristics We plan to implement other meta-heuristic algorithms (e.g. ACO, PSO, ...) to be able to perform experiments with other algorithm combinations.

Implementation of other problems Boolean Satisfiability problem and Job Shop Scheduling problem [13] are near to be implemented.

5.2.2 Works above the Problem Barrier

5.2.2.1 Parameter Setting Reasoning

The effectiveness of a meta-heuristic algorithm solving a given problem depends on parameters that are set. Values of the parameters are 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 (see section 3.3.2 for information about the role of the adviser agent).

Adviser Agent The adviser agent provides parameter settings to algorithm agents. We remark that algorithm agents ask the adviser agents for meta-heuristic parameters and send back reports describing meta-heuristic 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.

There are two possible policies of parameter settings–static and dynamic. Static policy gives the algorithm a set of parameter values at the beginning and the values are no changed during the algorithm runtime. Opposite to this, dynamic policy gives a new set of parameter values during the algorithm runtime, that reflect quality of current searching. Both types of policies are planned for future work.

Keeping the Global Population Diverse Enough The solution pool agent is responsible for providing candidate solutions to particular algorithm agent. It could maintain the population diverse enough by selecting candidate solution from the pool. This could be done, for example, by employing information from phenotype evaluator, which the solution pool agent could dispose.

Meta-Heuristic Competition Because of limited resources and possibly unlimited number of algorithm agents, we have to select which combination of meta-heuristics 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.

Bibliography

[1] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer (August 1999)

[2] Pinto, V.D., Pottenger, W.M., Thompkins, W..: A survey of optimization techniques being used in the field. In: In the Proceedings of the Third International Meeting on Research in Logistics, Quebec, Canada (2000)

[3] Andersson, J.: A survey of multiobjective optimization in engineering design johan (2000)

[4] Weise, T.: Global Optimization Algorithms Theory and Application. July 16, 2007 edn. Thomas Weise (July 2007) Online available at http://www.it-weise.de/projects/book.pdf.

[5] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology (2003)

[6] NEOS: Neos guide. `http://www-new.mcs.anl.gov/otc/Guide/OptWeb/opt.html`

[7] CSEP: Mathematical optimization. Computational Science Education Project. (1995)

[8] Russell, S.J., Norvig: Artificial Intelligence: A Modern Approach (Second Edition). Prentice Hall (2003)

[9] Voß, S.: Meta-heuristics: The state of the art. In: ECAI '00: Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers, London, UK, Springer-Verlag (2001) 1–23

[10] Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. **35**(3) (2003) 268–308

[11] Glover, F.W., Kochenberger, G.A.: Handbook of Metaheuristics (International Series in Operations Research & Management Science). Springer (January 2003)

[12] Yang, X.S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press, Frome, BA11 6TT, United Kingdom (2008)

[13] 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

[14] Freisleben, B.: Metaevolutionary approaches. In: Handbook of Evolutionary Computation, University Press (1997) C7.2

[15] Back, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK, UK (1997)

[16] Grefenstette, J.: Optimization of control parameters for genetic algorithms. IEEE Trans. Syst. Man Cybern. **16**(1) (1986) 122128

[17] Fogel, D., Fogel, L., Atmar, J.: Meta-evolutionary programming. In: Signals, Systems and Computers, 1991. 1991 Conference Record of the Twenty-Fifth Asilomar Conference on. (Nov 1991) 540545 vol.1

[18] Diosan, L.S., Oltean, M.: Evolving evolutionary algorithms using evolutionary algorithms. In: GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, New York, NY, USA, ACM (2007) 24422449

[19] Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. In: Parameter Setting in Evolutionary Algorithms. Springer (2007) 1946

[20] Deugo, D., Ferguson, D.: Evolution to the xtreme: Evolving evolutionary strategies using a meta-level approach. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Portland, Oregon, IEEE Press (20-23 June 2004) 3138

[21] Smith, J., Fogarty, T.: Operator and parameter adaptation in genetic algorithms. Soft Computing (1) (1997) 8187

[22] Moscato, P., Cotta, C.: Memetic algorithms. (2004) 53–85

[23] Hart, W., Krasnogor, N., Smith, J., eds.: Recent Advances in Memetic Algorithms. Springer, Berlin, Heidelberg, New York (2004)

[24] Ong, Y.S., Keane, A.J.: Meta-lamarckian learning in memetic algorithms. IEEE Trans. Evolutionary Computation **8**(2) (2004) 99110

[25] Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: LECTURE NOTES IN COMPUTER SCIENCE, Springer (2005) 41–53

[26] Talbi, E.G.: A taxonomy of hybrid metaheuristics. Journal of Heuristics **8**(5) (2002) 541–564

[27] Jourdan, L., Basseur, M., Talbi, E.G.: Hybridizing exact methods and metaheuristics: A taxonomy. European Journal of Operational Research **199**(3) (December 2009) 620–629

[28] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82

[29] Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. Intell. Data Anal. **12**(1) (2008) 3–23

[30] Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent developments. In Cotta, C., Sevaux, M., Srensen, K., eds.: Adaptive and Multilevel Metaheuristics. Volume 136 of Studies in Computational Intelligence. Springer (2008) 3–29

[31] Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: A classification of hyper-heuristic approaches, Nottingham, UK, University of Nottingham (2009)

[32] Cowling, P.I., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, London, UK, Springer-Verlag (2001) 176–190

[33] Fogel, D.B.: Phenotypes, genotypes, and operators in evolutionary computation. (1995) 193–198

[34] 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

[35] Roli, A., Milano, M.: Magma: A multiagent architecture for metaheuristics. IEEE Trans. on Systems, Man and Cybernetics - Part B **34** (2002) 2004

[36] Satoru Hiwa, T.H., Miki, M.: Hybrid optimization using direct, ga, and sqp for global exploration. In: IEEE Proceedings of 2007 Congress on Evolutionary Computation. (2007) 17091716

[37] Málek, R.: Alternation of meta-heuristic algorithms with runtime analysis and parameter adaptation (2007)

[38] Roli, A., Roli, A., Milano, M., Milano, M.: Metaheuristics: a multiagent perspective. Technical report, University of Bologna (Italy (2001)

[39] Özcan, E., Bilgin, B., Korkmaz", E.E.: Hill climbers and mutational heuristics in hyperheuristics. In: PPSN. (2006) 202211

[40] 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