Università degli Studi di Bologna
DEIS

MAGMA: A Multiagent Architecture for Metaheuristics

Andrea Roli and Michela Milano

August 28, 2002

DEIS Technical Report no. DEIS-LIA-02-007 *LIA Series no. 60*

MAGMA: A Multiagent Architecture for Metaheuristics

Andrea Roli and Michela Milano

DEIS
Università degli Studi di Bologna
Viale Risorgimento 2, 40136 Bologna (Italia)
`aroli,mmilano@deis.unibo.it`

August 28, 2002

Abstract. In this work we introduce a multiagent architecture conceived as a conceptual and practical framework for metaheuristic algorithms (MAGMA, MultiAGent Metaheuristics Architecture). Metaheuristics can be seen as the result of the interaction among different kinds of agents: level 0 agents constructing initial solutions, level-1 agents improving solutions and level-2 agents providing the high level strategy. In this framework, classical metaheuristic algorithms can be smoothly accommodated and extended, and new algorithms can be easily designed by defining which agents are involved and their interactions. Furthermore, with the introduction of a fourth level of agents, coordinating lower level agents, MAGMA can also describe, in a uniform way, cooperative search and, in general, any combination of metaheuristics.

We propose this perspective with the aim to achieve a better and clearer understanding of metaheuristics, obtain new algorithms and suggest guidelines for a software engineering-oriented implementation. Some specializations of the general architecture will be provided in order to show that existing metaheuristics can be easily described in our framework.

Keywords: *Metaheuristics, agents, combinatorial optimization*

Contents

1 Introduction

Metaheuristic techniques are used to solve hard problems (generally NP-hard problems) instead of, or in conjunction with, exact algorithms. When the problem size becomes very large, exact algorithms might be too computationally expensive. In these cases, approximate algorithms (which do not guarantee to find an optimal solution) are a very effective alternative to complete methods. Metaheuristics are approximate algorithms which encompass and combine constructive methods, local search strategies, local optima escaping strategies and population-based search. They include, but are not restricted to: Tabu Search [26], Simulated Annealing [38], Evolutionary Computation [8], Memetic Algorithms [45], Scatter Search [25], Ant Colony Optimization [17, 10], GRASP [19], Iterated Local Search [54, 53] and Variable Neighborhood Search [28]. Surveys and current research on metaheuristics can be found in [59, 10, 6].

In this paper, we revisit metaheuristics in a multiagent perspective and provide a uniform framework, MAGMA, where existing and newly designed metaheuristic algorithms can be smoothly accommodated. In Section 2 we survey the most commonly used classifications of metaheuristics. Section 3 defines basic concepts for the definition of metaheuristics. Then we describe MAGMA in Section 4 and provide a formal definition of the agents. In Section 5 we show some specializations of MAGMA that describe some among the best known metaheuristic algorithms. In Section 6 we discuss important themes of this approach and Section 7 outlines related works. Finally, we present some conclusions and open perspectives.

2 Metaheuristic algorithms classifications

Classifications of metaheuristics are useful when they help to outline similarities and differences among algorithms. The discriminant is the set of properties used to classify the algorithms. An important aspect for the classification is the level of abstraction at which the algorithms are described. Once some common key properties have been recognized, a general description can be formulated, and the algorithms of that class can be defined as specializations of it. A metaheuristics framework is useful for:

- comparing existing algorithms,
- analyzing algorithm properties and identifying their strengths and weaknesses,
- designing hybrid or brand new algorithms,
- supporting software engineering.

There are different ways to classify and describe metaheuristic algorithms. Perhaps, the most familiar and easiest is the description based on the algorithm origins. There are nature-inspired algorithms, like Genetic Algorithms and Ant Algorithms, and non nature-inspired ones like Tabu Search and Iterated Local Search. This classification is quite rough for two main reasons: *(i)* recent hybrid algorithms, e.g., memetic algorithms [45], do not completely fit neither class or, in a sense, they fit both at the same time, *(ii)* this classification is not helpful to compare algorithms.

Another way to classify metaheuristics is differentiating them between single and population heuristics. The first are commonly called *trajectory methods* and encompass local

Algorithm 1 Adaptive Memory Programming

Initialize the memory
while stopping condition not met **do**
  Generate a new provisory solution s using data stored in the memory
  Improve s with a local search; let s' be the improved solution
  Update the memory using pieces of knowledge brought by s'
end while

search-based metaheuristics, like Tabu Search, Iterated Local Search and Variable Neighborhood Search. They all share the property of describing a trajectory (continuous or not) in the search space during the search process. Population metaheuristics, conversely, generate search processes which dynamically describe the evolution of a set of points in the search space. The most important population based methods are Evolutionary Computation, Memetic Algorithms, Scatter Search and Ant Colony Optimization.

A very interesting framework is called *Adaptive Memory Programming* (AMP, [57]). AMP describes a class of metaheuristics as algorithms which use a memory to store information collected during the search process; this information is used to bias the solution construction and the future search (from that the term *adaptive*). AMP high level algorithm[1] is reported in Algorithm 1.

Within this general framework, several metaheuristics can be described, like Tabu Search, Ant Colony Optimization (ACO) and Scatter Search. Tabu Search explicitly handles a short term memory (the *tabu list*), which is used to avoid cycling among the same set of states. Population metaheuristics memory is implicitly embedded in shared data structures (population itself in evolutionary algorithms and pheromone trails in ACO). AMP clearly points out the importance of the use of memory during the solution construction and the search. Nevertheless, the level of abstraction chosen enables only to informally introduce concepts of memory, stored information and solutions, and its usefulness is limited to a general description of algorithms which make use of search history. AMP cannot be used as a framework which helps to design new algorithms, as its descriptive level is too general.

Other ways of describing metaheuristic algorithms have been proposed. For example, in [29] a framework for evolutionary computation algorithms is discussed, where evolutionary algorithms are described depending on seven main features: individual representation, evolution process, neighborhood structure, information sources, infeasibility, intensification and diversification.

In the next sections, we introduce some preliminary concepts like the problem model and the fitness landscape along with some examples. Then we define MAGMA, a multilevel multiagent architecture designed for metaheuristics. Finally, some specialization of the architecture on GRASP, Ant Colony Optimization, Iterated Local Search, Memetic algorithms and cooperative search are provided showing that a large class of metaheuristics can be fit in the proposed framework.

[1] As it appears in [57].

3 Problem, Models and Fitness Landscape

In this section we introduce the problem classes metaheuristics usually attack and we briefly discuss some design issues of problem modelization. Furthermore, we introduce the fitness landscape, which is a fundamental component of our definition of search process.

3.1 Search Problem Formalization

The widest class of problems metaheuristics deal with are Constraint Satisfaction Problems (CSPs) and Combinatorial Optimization Problems (COPs). They can be defined by:

- a set of variables, $X = \{x_1, \ldots, x_n\}$;
- variable domains, $D_i \ \forall i = 1, \ldots, n$;
- constraints among variables, $c_j(Y_j)$, $Y_j \subseteq X$, $\forall j = 1, \ldots, m$;
- an *objective function* to be optimized (minimized or maximized).

In CSPs the objective function is constant, as they are solved by any feasible solution, whilst for COPs is a non trivial function. There are important variants of these problems: MAX-CSPs [23], in which the objective is to maximize the number of satisfied constraints (or a weighted sum of them) and Multi-criteria COPs [61, 27] in which more than one criterion should be optimized.

A *model* is a non-ambiguous and formal definition of the problem: what are the main entities, the constraints, the costs (or values) associated with configurations of entities and the goal.

Often the objects defined in the model are directly used by the algorithm. The choice of a specific model is, in general, "algorithm-oriented"(see Figure 1), because it has to be tailored to the algorithm that will be used to solve the problem. This holds both for systematic constructive methods and metaheuristics. For example, a problem modeled via binary variables and linear inequalities will be effectively solved through Integer Programming techniques, while the same model is not in general the best one for a Constraint Programming finite-domain solver, even if it can be sometimes applied. In the metaheuristics context, if, for instance, simple genetic algorithms are used, solutions may be coded as binary strings. The model may also define an alternative objective function, for example introducing penalties for infeasible solutions, or using smoothing functions to shape the search landscape. Observe that the search process may also dynamically change models (e.g., by swapping between two different objective functions [4]).

3.2 Fitness Landscapes

The local search process can be viewed as an exploration of a landscape aimed at finding a global optimum, or, at least, a "good" local optimum.

A *Fitness Landscape* is defined by a triple:

$$\mathcal{L} = (S, \mathcal{N}, F)$$

where:

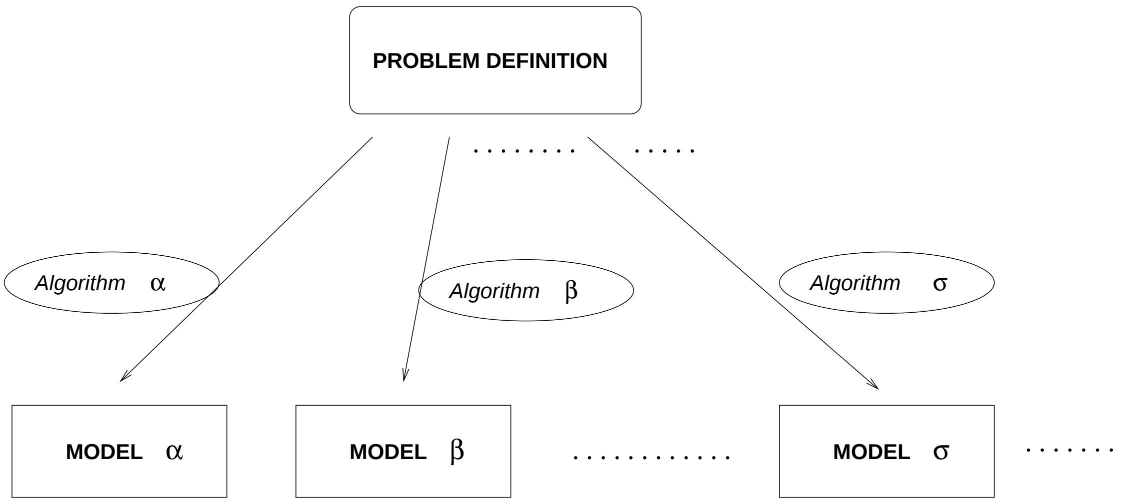

Figure 1. *Conceptual process from the original problem to the representation used in the algorithm. The model definition is algorithm dependent: solution space, neighborhood structure and objective function have to be tailored to the chosen metaheuristics.*

- S is the set of solutions (or states);
- $\mathcal{N}$ is the neighborhood function $\mathcal{N} : S \to 2^S$ that defines the neighborhood structure, by assigning to every $s \in S$ a set of states $\mathcal{N}(s) \subseteq S$.
- F is the objective function, in this specific case called *fitness function*, F: $S \to \mathbb{R}^+$.

The Fitness Landscape (hereafter referred to as FL) can be interpreted as a graph (see Figure 2) in which nodes are solutions (with their fitness value) and arcs represent the neighborhood relation between states.

The neighborhood function $\mathcal{N}$ implicitly defines an *operator* φ which takes a state s_1 and transforms it into another state $s_2 \in \mathcal{N}(s_1)$. Dually, given an operator φ, it is possible to define a neighborhood of a variable $s_1 \in S$:

$$\mathcal{N}_\varphi(s_1) = \{s_2 \in S \setminus \{s_1\} \mid s_2 \text{ can be obtained by one application of } \varphi \text{ on } s_1\}$$

Usually, the operator is *symmetric*: if s_1 is a neighbor of s_2 then s_2 is a neighbor of s_1. In a graph representation (like the one depicted in Figure 2) undirected arcs represent symmetric neighborhoods. A property the neighborhood structure should have is to allow a path from every pair of nodes (i.e., the graph is connected).

The notion of FL and neighborhood permits to frame local search algorithms as search processes in a graph. The search starts from an initial node and explores the graph moving from a node to one of its neighbors, until it reaches a termination condition.

There exists another definition of fitness landscape [34], which is more formal and general and can be used here as well. However, for the purpose of this contribution it is sufficient to know that this formal definition can deal with states (nodes of the search graph) composed of populations of solutions. The key point is the introduction of multi-sets, which are sets with possible repetitions of elements. A multi-set substitutes a single solution and the operator

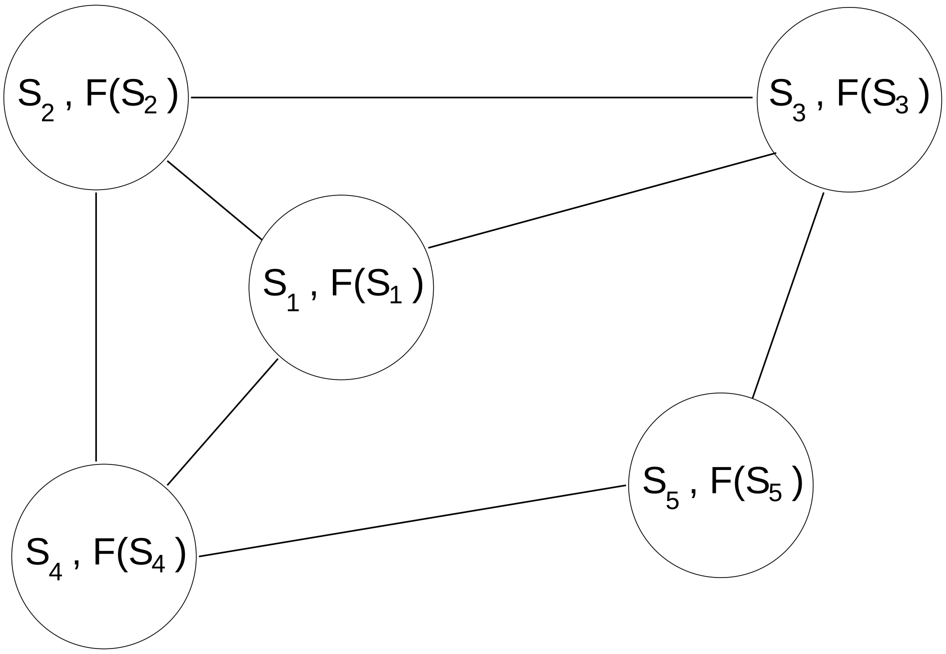

Figure 2. *Example of undirected graph representing a fitness landscape. Each node is associated with a solution s_i and its corresponding fitness value $F(s_i)$. Arcs represent transition between states by means of φ. Undirected arcs correspond to symmetric neighborhood structure.*

transforms a multi-set into another. Furthermore, the operator is defined as a function ψ : $M(S) \times M(S) \rightarrow [0, 1]$, which assigns a probability for each possible transition between states. This definition of FL enables to deal with population heuristics, like genetic algorithms, in the same way as simple local search algorithms.

There are some important design issues in developing a search algorithm over a FL: the solution representation, the neighborhood structure and the fitness function. Furthermore, there are some ways to cope with constraints. For example, it is possible to map a CSP or a COP into a Free Optimization Problem, where there are not constraints and infeasible solutions are penalized by the fitness function [18].

It is worth underlining that, given a fitness function, the choice of an operator determines the properties of the landscape. This is the "One Operator One Landscape" concept, introduced in [34, 35]. The algorithm performance is strongly affected by the model chosen and, in general, no best choice exists which leads to the best performance with every algorithm/problem combination[2].

3.3 Examples

In this section we present some examples of the previously introduced concepts.

[2]This empirical conjecture is theoretically supported by the *No Free Lunch Theorem* [62].

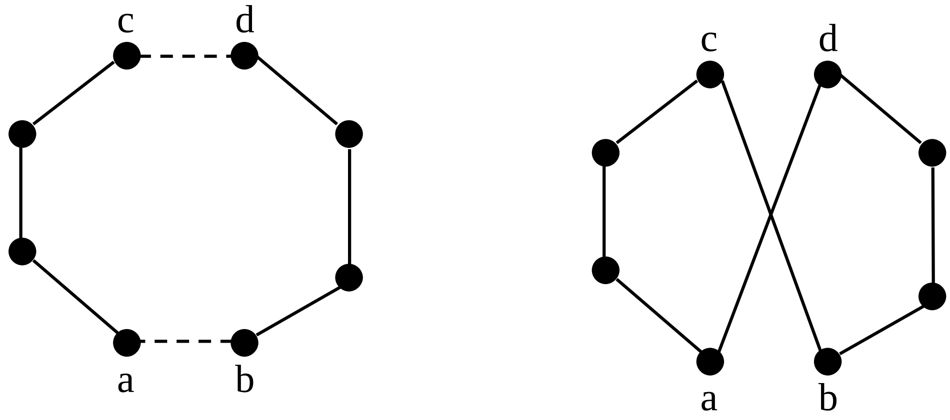

Figure 3. *Example of 2-exchange move. The tour can be reconstructed in only one way.*

3.3.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is defined as follows: given an undirected graph (supposed fully connected), with n nodes and each arc associated with a positive value, find the Hamiltonian tour with the minimum total cost. From the problem definition, several representations (models) are possible. We shall discuss two among the most common ones.

Representation 1

A solution (i.e., an Hamiltonian tour) can be represented as a sequence of n arcs (v_i, v_j), where v_i and v_j are nodes of the graph. The solution cost is given by the sum of costs of the arcs in the tour.

Two very successful neighborhood structures which can be introduced in the above-given representation are called 2-exchange and 3-exchange which lead to 2 and 3-opt improvement algorithms [33]. Both algorithms need to start with a feasible solution. The 2-exchange operator deletes two arcs from the feasible tour and reconnects the two paths into a tour in the other way (see Figure 3). In Figure 4 the 3-exchange move is sketched: three arcs are removed and the feasible tour is then reconstructed by using different arcs (in this case there are six possible ways to reconstruct the cycle).

The FL generated by the two moves are different, as the defined neighborhoods differ in size and connectivity properties. 2-exchange neighborhood $\mathcal{N}_{2-ex}$ has a cardinality proportional to n^2, while for 3-exchange $|\mathcal{N}_{3-ex}|$ is proportional to n^3. Observe that the solutions defined in this model are feasible for the original problem as well, because moves preserve the Hamiltonian tour structure.

Representation 2

A solution (i.e., an Hamiltonian tour) can be alternatively represented as a sequence of n values $\{v_1, v_2, \ldots, v_n\}$, where v_i are nodes of the graph and the solution represents the sequence of nodes in the tour. For example, $\{3, 5, 4, 2, 1\}$ represents the tour $3 \to 5 \to 4 \to 2 \to 1$. The solution cost is given by the sum of costs of the arcs in the tour.

A simple move operator may be the exchange of any two nodes in the sequence. If the initial solution is a feasible tour, the result of a move is guaranteed to be feasible. Similarly, it is possible to define moves which pick up more than 2 nodes and exchange them, leading to

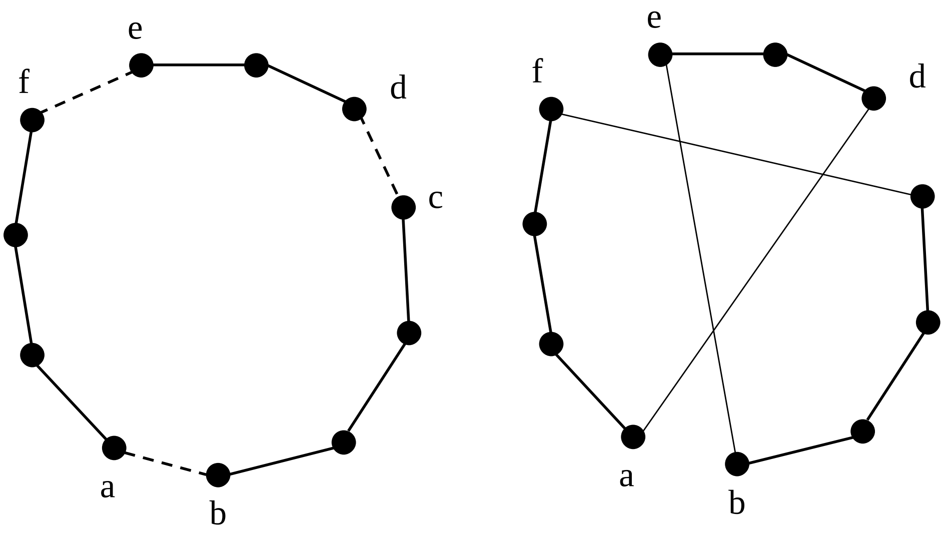

Figure 4. *Example of 3-exchange move. After the removal of three arcs there are six possible ways to reconstruct the tour. In this figure just one of them is shown.*

the definition of neighborhoods $\mathcal{N}_{k-perm}$. Observe that these operators define neighborhoods of cardinality proportional to $\frac{n!}{k!(n-k)!}$.

It is worth to note that, if the representation chosen $\{v_1, v_2, \ldots, v_n\}$ was interpreted as "the position of node i is v_i", we would have to introduce an additional constraint to avoid subtours.

3.3.2 Satisfiability Problem

In this example we discuss a possible representation of the Satisfiability Problem (SAT) when solved via Local Search.

SAT is defined as follows: given a set of clauses, each of which is the logical disjunction of k *literals* (a literal is a variable or its negation), the goal is to find an assignment to the variables that satisfies all the clauses. For example: the set of clauses $\{(x_1 \vee \neg x_2), (\neg x_1 \vee \neg x_3), (x_1 \vee x_3), (x_2 \vee \neg x_3)\}$ is satisfied by the assignments $\{(x_1 \leftarrow 1, x_2 \leftarrow 0, x_3 \leftarrow 0), ((x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0))\}$.

Let us suppose to solve SAT with local search. The SAT problem does not involve any optimization criteria since any solution satisfying all the constraints should be accepted. However, solving SAT with LS, we need to introduce in the model an objective function which evaluates the assignments with respect to their "closeness" to the satisfying ones. A typical objective function is the number of satisfied clauses, but other choices are possible[3]. Thus, feasible solution for the SAT problem are mapped in optimal solutions in the model.

4 MAGMA: MultiAGent Metaheuristic Architecture

The weak notion of *agent* states that an agent is a (software) system that enjoys the following properties [63, 60, 49]:

[3] See, for example, *non–oblivious functions* for MAXSAT problems in [5].

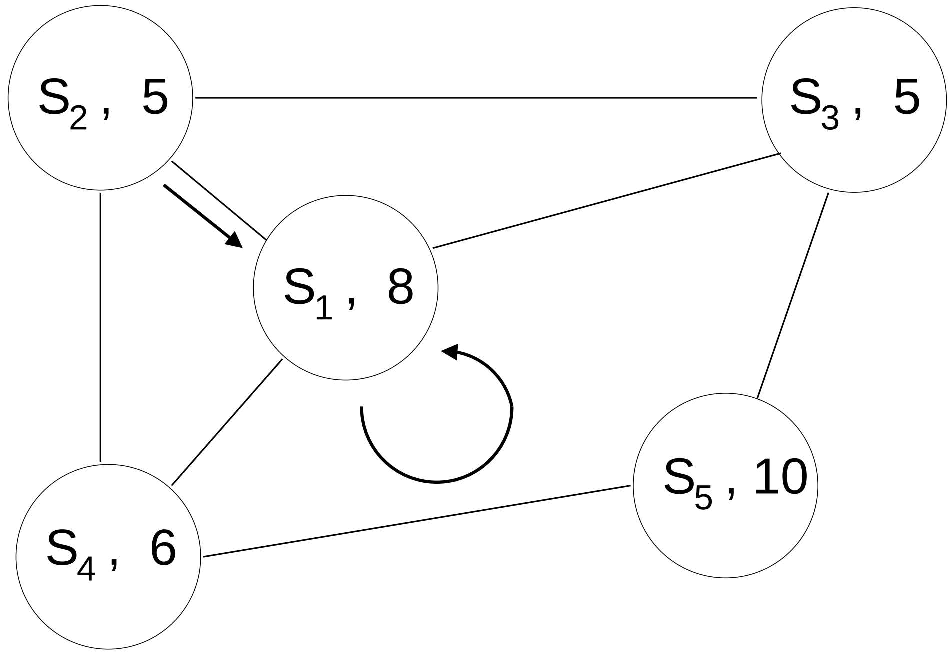

Figure 5. *An example of hill-climbing local search on a FL. If the agent starts at node s_2 it moves toward the best neighbor s_1 whose fitness value is 8. From node s_1 there are no other possibilities to move, because its neighbors have lower fitness. The agent has reached a local optimum and cannot reach the global optimum s_5.*

- *Autonomy*
- *Social ability*
- *Reactivity*
- *Pro-activeness*

In this context, we adopt the metaphor of agent, referring to a system able to build a solution, move over a landscape, communicate with other agents, be active (goal-oriented) and, possibly, be adaptive. In fact, we devise different kinds of agents, with different functionalities, perspectives and goals. These agents act in a multi-level architecture (MAGMA, depicted in Figure 6), composed of some levels each of which corresponds to a different level of *abstraction*.

At each level there are one or more specialized agents, each implementing an algorithm. The LEVEL–0 provides a feasible solution (or a set of solutions) for the upper level; it can be considered the *solution* level. LEVEL–1 deals with solution improvement and agents perform local search until a termination condition is verified; this can be defined as the level which deals with *neighborhood structure*. LEVEL–2 agents have a global view of the space, or, at least, their task is to guide the search toward promising regions. Therefore this can be defined as the *landscape* level. A fourth level, LEVEL–3, can be introduced to describe higher level strategies like cooperative search and any other combination of metaheuristics. Therefore this can be defined as the *coordination* level and deals with different landscapes and strategies.

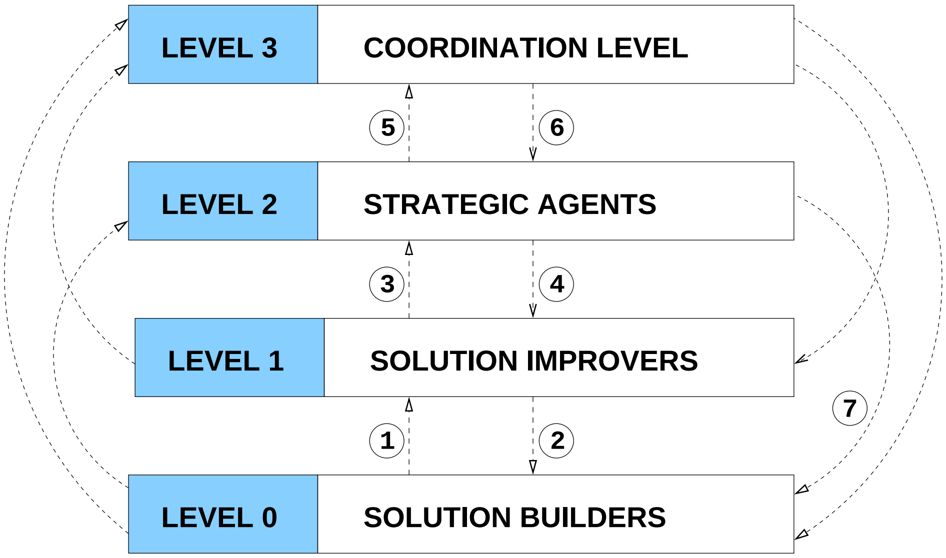

Figure 6. *Multi-level architecture for metaheuristic algorithms.*

Communications between any two levels are possible, therefore an algorithm can be described as the result of the interaction of some agents (algorithmic components) each specialized for a specific task. Two kinds of communication are possible: *horizontal* communication, which characterizes information exchange among agents of the same level and *vertical* communication, in which agents belonging to different levels exchange data. Communication among agents can be implemented through any kind of mechanism and protocol. The designer is free of choosing the most suitable communication scheme. For example, agents can use a blackboard or a message passing mechanism.

Agents can be formally defined with tuples, whose elements describes computational capabilities, knowledge and goals. This definition encompasses all the current metaheuristics and it can be further extended to include new metaheuristics. Each tuple is composed of two main components: the model $\mathcal{M}$ and the strategy $\mathcal{S}$. In the following, we formalize the tuples for each level, while in section 5 we will specialize the tuples on specific algorithms.

4.1 Definition of the multi-level architecture

LEVEL–0 To this level belong *solution builder* agents, whose task is to produce starting solutions. LEVEL–0 agents, hereinafter referred to as L0-agents, deal with problem entities like variables, domain values and constraints or partial solutions, depending on the representation chosen. The L0-agents' goal is to construct a starting solution for the upper level. L0-agents may exploit strategies like random initialization, greedy construction, probabilistic construction (e.g. pheromone based), etc. Their computational capabilities are usually quite limited. They are generally reactive agents, whose behavior is based upon a heuristic criterion (which may be dynamically changed by upper levels agents).

Agents of LEVEL–0 are defined by tuple $\mathcal{T}_0 = \langle \mathcal{M}_0, \mathcal{S}_0 \rangle$, where $\mathcal{M}_0 = \langle \mathcal{V}, \mathcal{D}, \mathcal{C}, \mathcal{OF} \rangle$. $\mathcal{V}$ are *components*, the domains $\mathcal{D}$ define the values that components can assume, $\mathcal{C}$ are the *constraints* and finally $\mathcal{OF}$ is the *objective function*. $\mathcal{S}_0$ is the *strategy* which will be specialized depending on the metaheuristic algorithm we are describing. We will provide some examples in Section 5. The *strategy* is the actual constructive algorithm, for instance it can be random, greedy, probabilistic, etc. *Components* can be variables or partial solutions, or any element which is used as a building block for solutions. *Constraints* are needed for the agent to construct solutions which are feasible with respect to the model (indeed, it is possible that they are infeasible for the original problem, as shown in Section 3.3.2 where an example of LS on SAT is discussed).

As output LEVEL–0 provides one or more solutions to the upper levels. In addition, LEVEL–0 can be triggered each time a restart is performed.

LEVEL–1 This level contains *solution improver* agents. Each agent searches in a Fitness Landscape (FL) with a local search, trying to improve the initial solution it got from the lower level. LEVEL–1 agents (L1-agents) implement various search algorithms and they can constitute either single independent search agents, or cooperating agents. L1-agents deal with solutions and neighborhood structures. The usual concept of *short term memory* belongs to this level: agents can use their recent past to intensify the search or to escape from local optima. Therefore, L1-agents are usually not just reactive, but also goal-oriented.

L1-agents are defined by tuple $\mathcal{T}_1 = \langle \mathcal{M}_1, \mathcal{S}_1 \rangle$, where $\mathcal{M}_1 = \langle Sol\ , \mathcal{N}, \mathcal{H}, \mathcal{F} \rangle$. $\mathcal{S}_1$ is the strategy used to improve the solution and again it will be specialized on the specific metaheuristic. Let us consider now the model $\mathcal{M}_1$: *Sol* is the *initial solution*, $\mathcal{N}$ is the *neighborhood structure*, $\mathcal{H}$ is the search *history* and $\mathcal{F}$ is the *fitness function*.

The *initial solution*, provided by LEVEL–0 (*arrow 1* in Figure 6) is the starting point of the search, guided by the *strategy* which, in general, makes use of the search *history* and the *fitness function*. A very important element is the *neighborhood structure*, which defines the portion of search space visible from each state. The neighborhood structure can be also dynamic (as for Tabu Search) or the agent can dynamically change neighborhood during the search process, if suggested by upper level agents (*arrow 4*).

The output of LEVEL–1 is the improved solution, which in general represents a local optimum or a state corresponding to search stagnation.

It is worth to stress that the more complex an agent is, the more difficult is the separation of basic algorithmic components. Therefore, instead of having LEVEL–1 agents with very sophisticated dynamic neighborhood structures, this architecture suggests to design simpler agents coordinated by an upper level agent. Thus, for example, metaheuristics with dynamic neighborhoods can be described as a system composed of LEVEL–1 agents with only one neighbor which alternate their search process under the coordination of a strategic agent of LEVEL–2.

LEVEL–2 LEVEL–2 agents (L2-agents) are *strategic* agents, i.e., they sample or abstract the search space looking for promising regions to explore.

Solutions produced by LEVEL–1 provide very useful information about the structure of optimal or suboptimal solutions. LEVEL–2 agents can

- store the best solutions found among those provided by LEVEL–1 (*arrow 3*);
- suggest what building blocks have the highest probability of being part of the optimal solution (*arrow 7*);
- suggest what regions of the search space are the most promising (*arrow 4*);
- dynamically bias the intensification/diversification balance (*arrows 4 and 7*);

- provide new building blocks for LEVEL–0 (*arrow 7*).

L2-agents deal with landscapes. *Long term memory* is implemented and exploited at this level, where data concerning the whole history of the search process of lower level agents are stored. One of the main uses of long term memory is the implementation of diversification strategies. L2-agents implement techniques like: direct comparison between solutions, statistics, positive feedback mechanisms, etc. They can also search in the abstract FL given by an approximation of the FL explored by L1-agents and return with guiding lights for the search at LEVEL–1 (*arrow 4*).

L2-agents are described by the tuple $\mathcal{T}_2 = \langle \mathcal{M}_2, \mathcal{S}_2 \rangle$, where $\mathcal{M}_2 = \langle \mathcal{P}, \mathcal{H}, \mathcal{F} \rangle$ and $\mathcal{S}_2$ is the strategy used for guiding lower level agents.

The model contains $\mathcal{P}$, i.e., the set of solutions of L1-agents, the *history* $\mathcal{H}$ and the *fitness function* $\mathcal{F}$. The set $\mathcal{P}$ of solutions provided by L1-agents is mainly used in population based methods, in which recombination and mutation operators are applied to the population. The *history* $\mathcal{H}$ is the structured ensemble of data collected during the overall search process. It includes information on every agent search process, like for instance the best solution found. This information is exploited by the *strategy* which leads lower level agents.

LEVEL–3 Agents belonging to this level (called L3-agents) coordinate lower level agents behavior. For example, cooperative search is implemented by these agents who know the model and strategy of lower level agents and coordinate their behaviours and communications. Moreover, L3-agents may evaluate the lower level agents results and reward agents which extracted the best solutions building blocks. They also could apply learning algorithms like evolutionary ones to select, adapt and improve the population of lower level agents (those actions involve communications expressed by *arrows 5* and 6). L3-agents perform meta-reasoning on the behavior of agents at lower levels. The computational capabilities of L3-agents can be the highest among all the levels, as they are learning and reasoning agents.

L3-agents are described by the tuple $\mathcal{T}_3 = \langle \mathcal{M}_3, \mathcal{S}_3 \rangle$, where $\mathcal{M}_3 = \langle \mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 \rangle$ and $\mathcal{S}_3$ is the strategy used for guiding lower level agents. The model $\mathcal{M}_3$ contains the complete description of lower level agents and the strategy $\mathcal{S}_3$ defines the specific coordination scheme.

It is worth noting that each level has the same structure composed by a model and a strategy. Observe that the higher the level, the higher the abstraction. In fact, $\mathcal{M}_0$ is strongly problem dependent: components represent problem entities and constraints are relations among them. On the contrary, $\mathcal{M}_1$ manipulates solutions independently from their semantics and $\mathcal{M}_2$ further enlarges the gap between the problem and its representation since it only focuses on fitness landscape analysis. Finally, $\mathcal{M}_3$ can be consider as a meta model containing the tuples of lower level agents.

5 Specializing MAGMA

MAGMA can be exploited either as a conceptual and implementation framework for metaheuristic algorithms, or as a metaphor for designing multiagent algorithms for optimization problems. In this section, we concentrate on the first topic and we show how some among the most effective metaheuristic algorithms can be described in terms of MAGMA agents. We will describe some specializations of the architecture showing that both trajectory and population based algorithms can be easily defined in our framework.

In the following sections, we will specialize the tuples describing the agents involved in each metaheuristic. Obviously, the models, i.e., $\mathcal{M}_i$ depend on the particular problem to be

Algorithm 2 Greedy Randomized Adaptive Procedure

```
while stopping criterion not satisfied do
    Construct a solution s with a Greedy Randomized procedure
    Apply local search to s
    Memorize the best solution found
end while
```

solved, thus here they are skipped. Thus, for each tuple, we will describe only the strategy.

There are metaheuristics which can just be described by L0-agents and L1-agents. For example, the basic Tabu Search can be realized with a single L0-agent which randomly constructs an initial solution and a single L1-agent which performs the search with one or more tabu lists. Of course, Tabu Search metaheuristics make often use of long term memory and therefore they need a further agent at LEVEL–2 which balances intensification and diversification, for instance by enlarging and restricting the tabu list.

The detailed description and discussion of the metaheuristic algorithms is beyond the scope of this paper and they are presented just in their easiest and general form. It is worth noting that variants of algorithms can be easily obtained by changing the algorithms implemented at any level. Moreover, it is always possible to introduce more than one agent per level. Communications between levels are very important as well, and in principle they involve all possible combinations of links between levels, even if the metaheuristic algorithms designed so far use just some of all possible communication links.

5.1 GRASP

We start our series of examples with a very simple algorithm: GRASP [19]. This algorithm is composed of a greedy construction procedure and a local search procedure (see Algorithm 2). The construction procedure builds a solution by applying a greedy function which, at each step, adds a randomly chosen component among the ones which have the highest greedy function value. Then, a local search phase starts; the choice of the local search algorithm depends on the problem to be solved. GRASP is easily described in MAGMA: the first level generates a solution by means of a greedy algorithm, a single L1-agent performs local search, and finally a L2-agent keeps track of the best solution found. Communications are unidirectional and the information flows from LEVEL–0 to LEVEL–1 and from LEVEL–1 to LEVEL–2. In Figure 7 the MAGMA version of GRASP is depicted.

The instantiations of the strategy in each tuple for GRASP are the following: $\mathcal{S}_0$ is a greedy random constructive strategy, $\mathcal{S}_1$ is a simple local search strategy implementing hill climbing, while $\mathcal{S}_2$ stores the current best solution.

5.2 Ant Colony Optimization

Ant Colony Optimization is a well known population-based metaheuristic for designing algorithms for COPs [15, 17, 16]. The inspiring idea behind ACO is the foraging behavior of real ants and the way they communicate. Real ants deposit a pheromone trail while they are walking and select their direction in a stochastic way. The higher the pheromone value on a path, the higher the probability the ant chooses that path. Artificial ants are simple agents that construct a solution by choosing the next component in a probabilistic way (function of the pheromone). In ACO, pheromone is a mean to communicate good solution components.

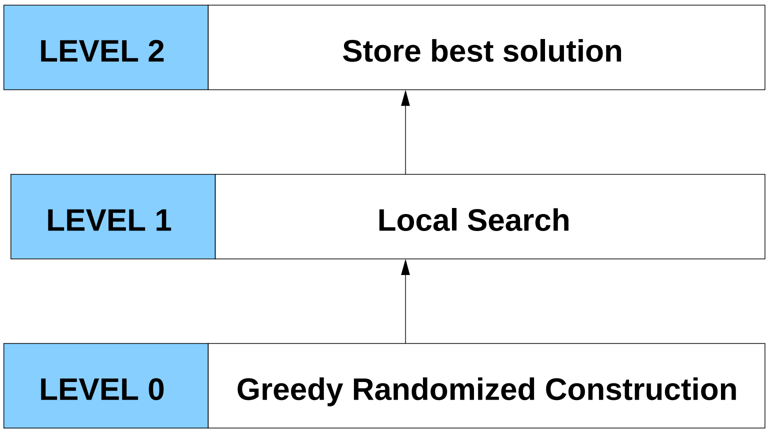

Figure 7. *The Multi-level architecture version of GRASP.*

Algorithm 3 Ant Colony Optimization

while stopping criterion not satisfied **do**
  Construct a set of solutions $S = \{s_1, s_2, \ldots, s_m\}$ by using pheromone trails and heuristic information
  Apply local search to every solution in S
  Update pheromone trails
end while

Several implementations of ACO have been proposed, often with different choices for the construction rules and the way to update pheromone trails [17, 24, 55, 56, 7, 48].

A generic ACO algorithm is described in Algorithm 3.

In Figure 8 is depicted the MAGMA specialization: L0-agents provide initial solutions by using a constructive procedure biased by pheromone trails. L1-agents perform local search and one L2-agent updates pheromone. The last information (new pheromone values) is then used again by L0-agents to construct new solutions. We observe that in this case the communication is also from the third level to the first and this means that the process is self-adapting, as a feedback loop involves the initial and the final steps.

The instantiations of the strategy in each tuple for ACO are the following: L0-agents implement a probabilistic constructive procedure $\mathcal{S}_0$, L1-agents implement any trajectory method $\mathcal{S}_1$, while one L2-agent stores the current best solution and update pheromone trails via $\mathcal{S}_2$.

5.3 Iterated Local Search

Iterated Local Search (ILS) is a simple but powerful metaheuristic algorithm [54, 53]. It applies local search (or a more general trajectory method) to an initial solution until it finds a local optimum; then it perturbs the solution and restarts local search from the perturbed state. The importance of the perturbation (change in the solution) is obvious: a very small change in the solution could not make it possible to escape from the local optimum basin of attraction; on the other side, a too strong modification can be comparable to a simple random restart. In order to accomplish these requirements, several criteria have been adopted, most of

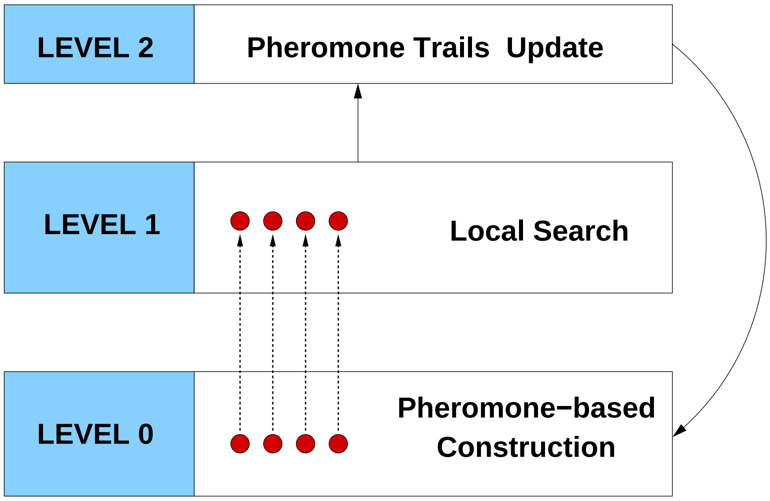

Figure 8. *The Multi-level architecture version of ACO with local search.*

Algorithm 4 Iterated Local Search

Generate initial solution s
Execute LS from an initial state s until a local optimum s^* is found
while stopping criterion not satisfied **do**
  Perturb s^* and obtain s'
  Execute LS from s' until it finds a local optimum $s^{*'}$
  On the basis of an *acceptance criterion* decide whether to set $s^* \leftarrow s^{*'}$
end while

which use the history of the search. A very simple version of ILS is described in Algorithm 4.

The design of ILS algorithms has several degrees of freedom in the choice of the initial solution, perturbation and acceptance criteria.

In the MAGMA framework, ILS can be described as follows (see Figure 9): one L0-agent provides an initial solution (either randomly, or heuristically generated), then it stops its activity. At LEVEL–1 one solution improver agent applies a local search algorithm to the solution and, when it meets a local optimum it stops, waiting for a new solution to improve. At LEVEL–2 the agent keeps track of the search process of the local search agent and, when it stops, it evaluates the new starting solution as a modification of the current one. Observe that, in this case, there is a continuous communication between LEVEL–1 and LEVEL–2, while LEVEL–0 participates just for the initialization, or for a random restart[4].

The instantiations of the strategy in each tuple for ILS are the following: a L0-agent implements a (randomly, or heuristically generated) constructive procedure $\mathcal{S}_0$, a L1-agent implements any trajectory method $\mathcal{S}_1$, while one LEVEL–2 agent stores the current best solution and perturbs the solution returned be the L1-agent via $\mathcal{S}_2$.

[4]Random restart can, of course, be added to every approximate algorithm.

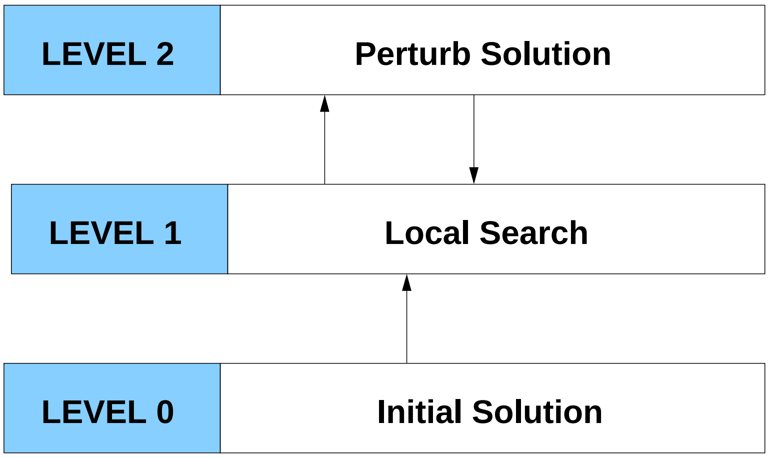

Figure 9. *The Multi-level architecture version of ILS.*

Algorithm 5 Memetic Algorithm

Generate initial population S
while stopping criterion not satisfied **do**
  Apply local search to every individual $s \in S$
  Apply cooperation operators to S and obtain S'
  Apply competition operators to S' and obtain S''
  $S \leftarrow S''$
end while

5.4 Memetic Algorithms

In this section, we propose the application of the MAGMA framework to Memetic Algorithms (MAs) [45]. We include in this category genetic algorithms which apply also local search, in a way similar to ACO plus local search. In a MA (see Algorithm 5), an initial population of solutions is generated; then every solution is improved by applying local search. The resulting new population cooperates and/or competes to produce a new population. The cooperation is a way to exchange information among individuals and it can be implemented as genetic operators like mating and crossover; the competition can be implemented with the selection genetic operator or more elaborated mechanisms.

A MA in MAGMA is described as follows: many L0-agents generate the initial population of solutions. Each solution can be generated either randomly, or by means of a constructive procedure. L1-agents take the solutions from the lower level and improve them by using local search (either some steps of a local search algorithm, or until they reach a local optimum). At LEVEL–2 we have one agent whose task is to generate new solutions by applying recombination and mutation operators. Then the new population is produced by applying cooperation or competition operators. The new population is directly given as input to LEVEL–1. Figure 10 shows the multi-level architecture for a generic MA.

The instantiations of the strategy in each tuple for MA has some similarities (at least for the first two levels) with ILS, with more than one agent per level. In MA many L0-agents implement a (randomly, or heuristically generated) constructive procedure $\mathcal{S}_0$, many L1-agent

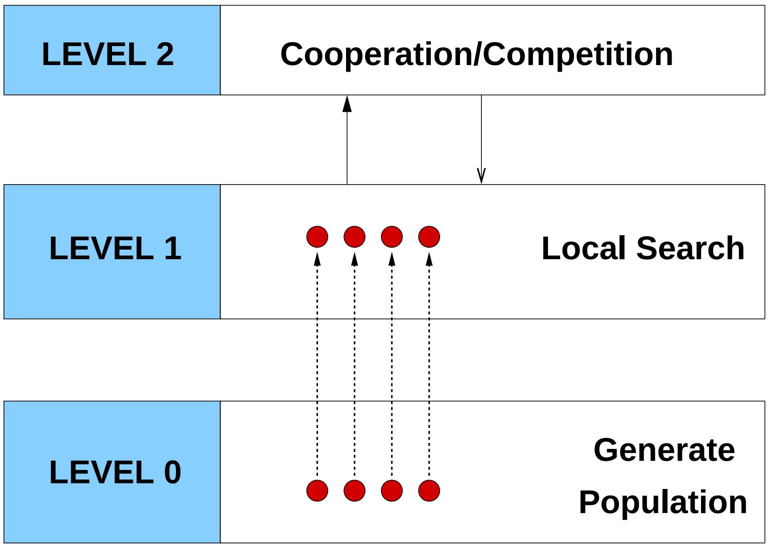

Figure 10. *The Multi-level architecture version of Memetic Algorithms.*

implements a local search $\mathcal{S}_1$, while one or more L2-agents evaluate the population and apply recombination, mutation and selection/competition operators via $\mathcal{S}_2$.

5.5 Cooperative Search

When a landscape is explored by means of more than one strategy with possibly different perspectives and information are exchanged among strategies, we obtain the so called cooperative search. Empirical results (see, for instance, [22]) show that some algorithms (agents) perform better than others on particular kinds of problems, while they are outperformed on other problems. It is conjectured that this depends upon the agent ability to exploit the fitness landscape characteristics. Among such properties are: ruggedness, number of local optima, distribution of local optima and topology of the basins of attraction [42, 51, 52, 36, 31, 32]. The choice of fitness function and operator defines such characteristics and it is reasonable that the combination of searches on different fitness landscapes (derived from the same original problem) can smooth the problem hardness (or, at least, effectively face it).

Cooperative search [30, 3, 12, 58, 50] consists of a search performed by agents which exchange information about states, solutions or other search space characteristics. Agents can be either homogeneous (implementing the same algorithm) or heterogeneous (different algorithms). Cooperative search has two main advantages:

- Different operators (i.e. different landscapes) can generate different projections of the search space.
- Different representations extract and show different problem properties.

Agents can exchange complete or incomplete solutions, information about promising regions to explore and about not promising regions to avoid. The last two are example of (respectively) *positive information* and *negative information* [12].

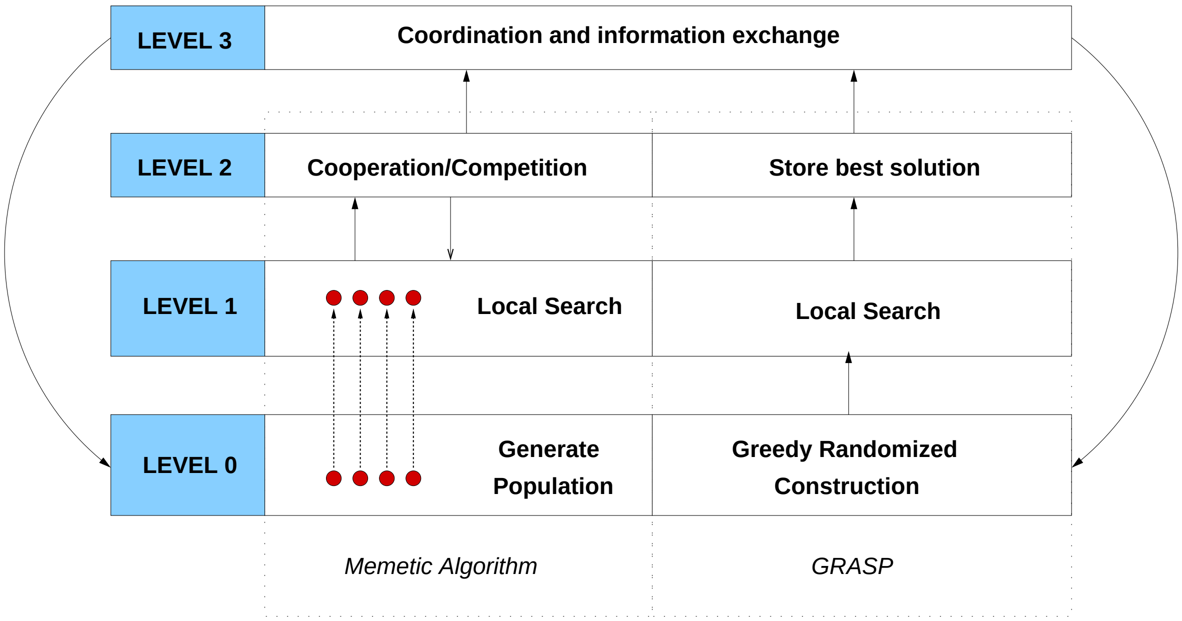

Figure 11. *The Multi-level architecture version of Cooperative Search.*

The usefulness of the multiagent metaphor emerges when considering design issues like: synchronization, communication, information filtering and implementation. A meaningful example of cooperative search in a multiagent system is reported in [12]: in addition to search agents, there are *referees* agents whose work is to filter information. This architecture permits to develop separately search agents (focusing on search algorithms), agent which collect and filter (evaluate) information, communication mechanisms and policies.

It is important to mention that cooperative search algorithms (systems) show their effectiveness and efficiency even if they are sequentially implemented. This means that improvements arise from information exchange and not only from the parallel implementation [41, 37].

In the following, we provide an example of cooperative search and its description in MAGMA. Suppose we have a system composed of a Memetic Algorithm (MA) collaborating with GRASP. Both metaheuristics have been already described in the MAGMA architecture respectively in sections 5.4 and 5.1. The aim of cooperative search is to let the two metaheuristics communicate and exchange results. In particular, we suppose that in every new generation of MA or whenever MA needs diversification, we insert the k best solutions found by GRASP. Vice versa, suggestions for building the restricted candidate list of GRASP are derived by the best individuals of the MA.

L0, L1 and L2-agents, are those described in the previous sections for MA and GRASP. These tuples associated to these agents represent the model of an L3-agent who supervises and coordinates the information exchange. The strategy $\mathcal{S}_3$ defines the frequency of the solution exchange, builds the new population for MA and the restricted candidate list for GRASP (see figure 11).

6 Discussion

In MAGMA, all direct communications between levels are allowed. In the previous examples, only some among the possible communication have been exploited and many possibilities to combine and design new algorithms have to be investigated. The explicit presence of levels permits to clearly identify information flows. For example, algorithms which have a loop LEVEL–0 $\rightarrow$ LEVEL–1 $\rightarrow$ LEVEL–2 $\rightarrow$ LEVEL–0 exhibit a feedback loop. Feedback loops characterize self-adapting systems and emerging phenomena, e.g., ACO dynamics.

Another point has to be underlined: different agents can coexist and work concurrently at the same level, like for example in Cooperative Search. They can use different models, but communicate by means of agents which filter and translate information. They perform a kind of *horizontal cooperation*, which can be achieved through higher level agents. For example, in cooperative search two solutions can be (horizontally) exchanged between two L1-agents thanks to level 3.

One of the key features of MAGMA is LEVEL–2. Generally, solutions are just implicitly analyzed (e.g., genetic algorithms, ant algorithms, etc.) or with *ad hoc* algorithms (e.g., vocabulary building, scatter search, etc.). L2–agents provide an explicit way to combine good building blocks and to know promising regions to explore further. Using the typical metaheuristics terminology, LEVEL–2 implements the actual search strategy, as it guides the lower level search and adjusts the balance between intensification and diversification.

An interesting aspect of MAGMA concerns the possibility of introducing *meta-reasoning* in metaheuristics through L3-agents which apply, beside cooperative search, for instance, learning techniques (clustering and information extraction, reinforcement learning, etc.). Another very interesting possibility is to use LEVEL–3 to perform resource management for the lower levels. For instance, a L3-agent can decide the priorities among L2-agents and allot more or less CPU time to agents depending on the requirements. This strategy could be useful in case of anytime algorithms, where as soon as a solution is required, it has to be provided. L3-agents can also select lower level agents as a function of the quality of the solutions provided.

Another relevant issue concerns the agent synchronization. In the specializations we have discussed, the easiest way to implement the algorithms is in a sequential and synchronous way: LEVEL–0 produces a solution, then LEVEL–1 improves it, and finally LEVEL–2 takes some decision depending on the search strategy. However, in the MAGMA architecture, and especially with distributed systems, this choice is restrictive. There is the possibility to implement asynchronous algorithms and provide synchronization mechanisms to the agents exploiting up-to-date technology for communication and synchronization. Beside the advantage of speeding up the overall computation, asynchrony introduces more robustness and scalability.

7 Related work

A multiagent perspective of metaheuristics has relations both with object-oriented frameworks for the algorithms implementation and Distributed Artificial Intelligence (DAI). In this section we briefly discuss these issues.

Although MAGMA has been presented as a conceptual framework for metaheuristics, it can also be used as a framework for the algorithms implementation. In the literature, some implementation frameworks for local search-based metaheuristics have been introduced [39, 43, 13, 14, 20, 2, 11]. They all share the same aims: on the one hand, fast prototyping and implementation, on the other hand, systematic comparison of algorithms. Different are the implementation choices, but most of them provide a basic structure of two levels[5]: construction of the initial solution and further improvement. In [13, 14] an interesting hierarchical structure of object-oriented classes goes into the direction of MAGMA (even though restricted to local search-based metaheuristics). The authors define four levels:

1. Basic components (I/O, state, moves)
2. Local Search problem elements (State manager, neighborhood explorer)
3. Local search-based metaheuristics (Iterative Improvement, Tabu Search, Simulated Annealing, etc.)
4. Local Search solving strategy (Simple solver, Token-ring solver, etc.)

The fourth level can be interpreted as our LEVEL–2, as it guides the search performed by the third level which corresponds to our LEVEL–1. However, the fourth level just represents the outermost cycle of the algorithms and does not implement the search strategy.

Note that the model component here are put altogether in the first level, while in MAGMA we allow to have different models/components in different levels.

Research on DAI and Multiagent Systems covers several theoretical and applicative topics. The relations with the multiagent approach for metaheuristics emerge mainly in *distributed search*, where agents cooperate to solve a search problem [60, 47, 40]. The basic approach consists of subdividing the search space, assigning to each agent a subspace and let each agent "locally" search in its own subspace. Protocols and algorithms have been developed to maintain consistency and to combine the solutions provided by the agents into a complete solution. Several cooperative techniques have also been proposed and studied, beside different synchronization methods. Even if distributed search deals with distributed implementation of classical complete search algorithms, concepts and researches about the formalization of communication and cooperation among (heterogeneous) agents may be effectively included in the specializations and implementations of MAGMA.

8 Conclusion and Future work

In this paper, we have discussed a multiagent approach to metaheuristics. We have described the multi-level architecture MAGMA, where agents are characterized by different levels of abstraction and objectives. One of the advantages of using MAGMA as a framework for metaheuristic algorithms is the easy way of comparing and varying algorithms. Moreover, we are not restricted to the one level - one algorithm paradigm, since we can create some agents in each level and try to exploit the power of cooperation. Another advantage is that the multiagent metaphor gives direct suggestions for the algorithm implementation. Sequential and parallel (synchronous and asynchronous) implementations can be easily designed. In our opinion the MAGMA framework for metaheuristics clearly shows that these algorithms are more than simple local optima escaping strategies. They use enhanced intelligent strategies

[5] As it can be expressed with our framework terminology.

to explore the search space, from the learning mechanisms of ACO, to advanced trajectory guidance in ILS.

Future work covers the following topics:

- Design new hybrid algorithms.
- Explore the possibilities of cooperative search.
- Use of complex system metaphor; in particular, since the information flow is explicitly traced in MAGMA, a characterization of emergence in metaheuristics algorithms can be developed.

Acknowledgments

We would like to thank Paolo Torroni for his helpful comments and suggestions on this work. This work was supported by the SOCS project, funded by the CEC, contract IST-2001-32530. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* Wiley-Interscience, 1997.

[2] A.A. Andreatta, S.E.R. Carvalho, and C.C. Ribeiro. An object-oriented framework for local search heuristics. In *Proceedings of the 26th Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA '98)*, pages 33–45. IEEE, Piscataway, 1998.

[3] V. Bachelet and E.G. Talbi. Cosearch: a co-evolutionary metaheuristics. In *Proceedings of Congress on Evolutionary Computation – CEC'2000*, pages 1550–1557, 2000.

[4] R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 1997.

[5] R. Battiti and M. Protasi. Solving MAX–SAT with non–oblivious functions and history–based heuristics. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, number 35 in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, pages 649–667. American Mathematical Society, Association for Computing Machinery, 1997.

[6] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. Technical Report IRIDIA/01-13 Université Libre de Bruxelles, Belgium, 2001.

[7] C. Blum, A. Roli, and M. Dorigo. HC–ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 399–403, Porto, Portugal, 2001.

[8] P. Calegari, G. Coray, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.

[9] I. Charon and O. Hudry. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135:86–101, 2001.

[10] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization.* Advanced topics in computer science series. McGraw-Hill, 1999.

[11] B. De Backer, V. Furnon, and P. Shaw. An object model for meta-heuristic search in constraint programming. In *Proceedings of CP–AI–OR'99–Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, 1999.

[12] J. Denzinger and T. Offerman. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of Congress on Evolutionary Computation – CEC'1999*, pages 2317–2324, 1999.

[13] L. Di Gaspero and A. Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. Technical report UDMI/13/2000/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2000.

[14] L. Di Gaspero and A. Schaerf. EasyLocal++: An object-oriented framework for the design of local search algorithms and metaheuristics. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 1, pages 287–292, Porto – Portugal, 2001.

[15] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999. Also available as Technical Report IRIDIA/99-1, Université Libre de Bruxelles, Belgium.

[16] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Art. Life*, 5(2):137–172, 1999.

[17] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[18] A.E. Eiben and Z. Ruttkay. Constraint satisfaction problems. In T. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Computation.* IOP Publishing Ltd. and Oxford University Press, 1997.

[19] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[20] A. Fink and S. Voss. Reusable metaheuristic software components and their application via software generators. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 637–641, Porto – Portugal, 2001.

[21] F. Focacci and F. Laburthe and A. Lodi. Local Search and Constraint Programming. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 451–454, Porto – Portugal, 2001.

[22] C. Fonlupt, D. Robilliard, P. Preux, and E.G. Talbi. Fitness landscapes and performance of meta-heuristics. In Stefan Voss, Silvano Martello, Ibrahim Osman, and Catherine Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*. Kluwer Academic, 1999.

[23] E. Freuder and R. Wallace. Partial Constraint Satisfaction. Artificial Intelligence, 58:21–70, 1992.

[24] L.M. Gambardella, É.D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50:167–176, 1999.

[25] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, Advanced topics in computer science series. McGraw-Hill, 1999.

[26] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[27] M. Hansen. Tabu search for multiobjective optimization: MOTS. 13th International Conference on Multiple Criteria Decision Making, 1997.

[28] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In Stefan Voss, Silvano Martello, Ibrahim Osman, and Catherine Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, chapter 30, pages 433–458. Kluwer Academic Publishers, 1999.

[29] A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.

[30] T. Hogg and C.P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of AAAI93*, pages 213–235. AAAI Press, 1993.

[31] W. Hordijk. A measure of landscapes. *Evolutionary Computation*, 4(4), 1996.

[32] W. Hordijk and P.F. Stadler. Amplitude spectra of fitness landscapes. *Adv. Complex Systems*, 1(1), 1998. Also available as SFI working paper 98-02-021.

[33] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: a case study. In E.H. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[34] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, Univ. of New Mexico, Albuquerque, NM, 1995.

[35] T. Jones. One operator, one landscape. Santa Fe Institute Technical Report 95-02-025, Santa Fe Institute, 1995.

[36] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th Int. Conference on Genetic Algorithms*, pages 184–192. Kaufman, 1995.

[37] S.A. Kauffman and W. Macready. Technological evolution and adaptive organizations. *Complexity*, 26(2):26–43, March 1995.

[38] S. Kirkpartick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science, 13 May 1983*, 220(4598):671–680, 1983.

[39] F. Laburthe and Y. Caseau. SALSA: A language for search algorithms. In M. Maher and J.F. Puget, editors, *Principle and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 1998.

[40] V.R. Lesser. An overview of DAI: Viewing distributed AI as distributed search. *Journal of Japanese Society for Artificial Intelligence-Special Issue on Distributed Artificial Intelligence*, 5(4):392–400, 1990.

[41] W.G. Macready and S.A. Kauffman A.G. Siapas. Criticality and parallelism in combinatorial optimization. *Science*, 271:56–59, January 1996.

[42] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced topics in computer science series. McGraw-Hill, 1999.

[43] L. Michael and P. van Hentenryck. Localizer: A modeling language for local search. *INFORMS Journal of Computing*, 11:1–14, 1999.

[44] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms. Tech. Rep. Caltech Concurrent Computation Program 826, California Institute of Technology,Pasadena, California, USA, 1989.

[45] P. Moscato. Memetic algorithms: A short introduction. In F. Glover D. Corne and M. Dorigo, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

[46] G. Pesant and M. Gendreau. A View of Local Search in Constraint Programming. In E.C. Freuder, editor, *Principles and Practice of Constraint Programming - CP'96*. LNCS 1118, pages 353–366. Springer-Verlag, Berlin Heidelberg, 1996.

[47] N.V.N. Prasad, S.E. Lander, and V.R. Lesser. Cooperative learning over composite search spaces: Experiences with a multi-agent design system. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 1, pages 68–73. AAAI Press/MIT Press, 1996.

[48] A. Roli, C. Blum, and M. Dorigo. ACO for maximal constraint satisfaction problems. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 1, pages 187–191, Porto – Portugal, 2001.

[49] S.J. Russell and P. Norvig. *Artificial Intelligence. A modern approach*. Simon & Schuster Company, 1995.

[50] L. Sondergeld and S. Voss. Cooperative intelligent search using adaptive memory techniques. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, chapter 21, pages 297–312. Kluwer Academic Publishers, 1999.

[51] P.F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, volume 461 of *Lecture Notes in Physics*, pages 77–163, Berlin, New York, 1995. Springer Verlag. SFI preprint 95-03-030.

[52] P.F. Stadler. Landscapes and their correlation functions. *J. Math. Chem.*, 20:1–45, 1996. SFI preprint 95-07-067.

[53] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report aida-99-03, FG Intellektik, TU Darmstadt, 1999.

[54] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications.* DISKI - Dissertationen zur Künstliken Intelligenz. infix, 1999.

[55] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw-Hill, 1999.

[56] T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[57] E.D. Taillard, L.M. Gambardella, M. Gendreau, and J. Potvin. Adaptive memory programming: A unified view of meta-heuristics. European Journal of Operational Research, 135:1–16, 2001.

[58] M. Toulouse, T.G. Crainic, and B. Sansò. An experimental study of the systemic behavior of cooperative search algorithms. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, chapter 26, pages 373–392. Kluwer Academic Publishers, 1999.

[59] S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, 1999.

[60] G. Weiss, editor. *Multiagent Systems. A modern approach to distributed artificial intelligence.* The MIT Press, 2000.

[61] Luc N. Van Wassenhove and Ludo F. Gelders. Solving a bicriterion scheduling problem. European Journal of Operational Research, 4(1):42–48, 1980.

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

[63] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.