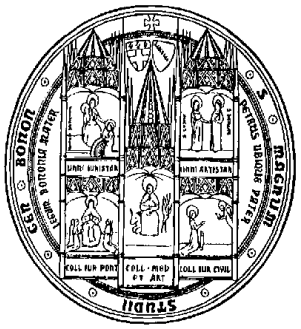

Università degli Studi di Bologna
DEIS

Metaheuristics: a Multiagent Perspective

Andrea Roli and Michela Milano

October 16, 2001

DEIS Technical Report no. DEIS-LIA-01-006 *LIA Series no. 50*

Metaheuristics: a Multiagent Perspective

Andrea Roli [1] and Michela Milano [1]

[1] *DEIS, Università di Bologna*
Viale Risorgimento, 2 - Bologna (Italy)
{aroli,mmilano}@deis.unibo.it

October 16, 2001

Keywords: *metaheuristics, combinatorial optimization, fitness landscape, agents*

Contents

1 Introduction **3**

2 Metaheuristics algorithms classifications **3**

3 Search Agents on Fitness Landscapes **4**
3.1 Search Problem Formalization 4
3.1.1 Example 1: Local Search for CSP 6
3.1.2 Example 2: The Ant Colony Metaheuristic 7
3.2 Fitness Landscapes 7
3.2.1 Example: Traveling Salesman Problem 9
3.3 Search Agents 10
3.4 Cooperative Search 12

4 A multi-level architecture **13**
4.1 Definition of the multi-level architecture 13

5 MLA and metaheuristic algorithms **14**
5.1 GRASP 15
5.2 Ant Colony Optimization 15
5.3 Iterated Local Search 16
5.4 Memetic Algorithms 17

6 Discussion **18**

7 Related work **19**

8 Conclusion and Future work **20**

Abstract

In this work we introduce the multiagent metaphor as a framework for metaheuristic algorithms. The first step is the definition of agents which search on a fitness landscape. In this framework it is possible to implement the classical metaheuristic algorithms and introduce cooperative search in a well-defined way. The second step extends the model in a multi–level system, where agents at each level have different computational capabilities and tasks: solution construction, solution improvement, extracting solutions building blocks, analyzing regions of the search space and perform meta-reasoning on the behavior of lower level agents. We propose this perspective with the aim to achieve a better and clearer understanding of metaheuristics, obtain new algorithms and suggest directions for a software engineering-oriented implementation.

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 dimension becomes very large, exact algorithms may not be useful since they are computationally too expensive. In these cases, approximate algorithms (which do not guarantee to find an optimal solution) are a very effective alternative. Metaheuristics are approximate algorithms which encompass and combine constructive methods, local search strategies, local optima escaping strategies and population-based search. They include, but they are not restricted to: Tabu Search [21], Simulated Annealing [32], Evolutionary Computation [5], Memetic Algorithms [39], Scatter Search [20], Iterative Improvement (Hill climbing et similia), Ant Colony optimization [14, 7], GRASP [16], Iterated Local Search [47, 46] and Variable Neighborhood Search [22]. Surveys and current research on metaheuristics can be found in [52, 7].

In this paper, we present a multiagent perspective of metaheuristics. In Section 2 we survey the most commonly used classification of metaheuristics. Section 3 defines the fitness landscape and describes the basic metaphor of search agents on fitness landscapes. Then we describe the multi–level architecture in Section 4. In Section 5 we show how metaheuristic algorithms can be described in the multi–level architecture. In Section 6 we discuss important themes of this approach and Section 7 outlines related works. Finally, we present some conclusions and ideas for future work.

2 Metaheuristics algorithms classifications

There are different ways to classify and describe metaheuristic algorithms. Perhaps, the most familiar and easy 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. The first reason is that recent hybrid algorithms do not completely fit neither class (or, in a sense, they fit both at the same time). The second reason is that this classification is not helpful in the comparison of algorithms.

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

Classifications of metaheuristics are useful when they help to outline similarities and differences among the algorithms. The discriminant is the set of properties used to classify and, very important, the level of abstraction at which the algorithms are described. Once some common key properties have been recognized among algorithms, a general description can be formulated, and the algorithms of that class can be defined as specializations of it. A very interesting such a framework is called *Adaptive Memory Programming* (AMP,[50]). 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 this the term *adaptive*). AMP may be sketched as follows[1]:

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

Within this framework several metaheuristics may be described, like Tabu Search, Ant Colony Optimization and Scatter Search. Tabu Search explicitly handles a short term memory (the *tabu list*), which is used to avoid cycling and visiting the same states again. Population metaheuristics memory is implicitly embedded in the population. AMP clearly points out the importance of the use of memory during the search and the solution construction. Nevertheless, the level of abstraction chosen allows 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 can not 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 [23] a framework for evolutionary computation algorithms is discussed. Evolutionary algorithms can be classified depending on the implementation of seven main features: individual representation, evolution process, neighborhood structure, information sources, infeasibility, intensification and diversification.

Every general description is an abstraction of the algorithms and focuses on the properties considered more important. A framework is useful if it allows to analyze and understand the algorithms, or if it suggests new algorithms as combinations of algorithmic components. In the following section we present a framework with the above-mentioned aims.

3 Search Agents on Fitness Landscapes

In this section we introduce a metaphor we consider useful for understanding, designing and implementing metaheuristic algorithms. First of all it is necessary a definition of the problems under exam, then we introduce the search on fitness landscapes. After that, we discuss in detail the multiagent metaphor.

3.1 Search Problem Formalization

Given a problem to solve, we generally define a *model* of it. A *model* is whatever non-ambiguous and formal definition of the problem: what are the main entities, the con-

[1] As it appears in [50].

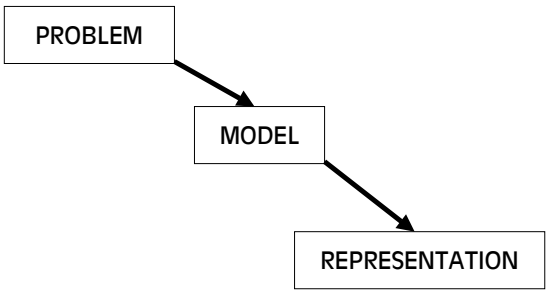

Figure 1. *Conceptual process from the original problem to the representation used in the algorithm.*

straints, the costs (or values) associated with configurations of entities and the goal[2]. The most important problems heuristic search deals with are Constraint Satisfaction Problems (CSPs), Combinatorial Optimization Problems (COPs). All of them can be defined by:

- a set of variables, $X = \{x_1, \ldots, x_n\}$, $n \in \mathbb{N}$;
- variable domains, $x_i : Dom_i \; \forall i = 1, \ldots, n, |Dom_i| < \infty$;
- constraints among variables, $c_j(x_{j_1}, \ldots, x_{j_k})$, $\forall j = 1, \ldots, m$, $k \in [1, \ldots, n]$;
- 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.

Often the objects defined in the model are directly used by the algorithm. For example, a problem modeled with Integer Linear Programming can be directly solved with *branch & bound* or heuristic constructive algorithms. Nevertheless, the model is, in general, independent of the search strategy chosen, therefore a *representation* has to be defined (see Figure 1). This choice is "algorithmic-oriented", because the representation is defined to tune the model with the algorithm that will be used to solve the problem. For example, if simple genetic algorithms are used, solutions may be coded as binary strings. The representation may also define an alternative objective function, for example introducing penalties for infeasible solutions, or using smoothing functions to shape the search landscape. Moreover, additional constraints may be added in the representation in order to deal with infeasibility. Observe that a search process may also dynamically change representation.

Formally, the representation defines the set of *solutions* (states) S, the *fitness function* and possibly additional constraints. Solutions correspond to the (feasible) configurations of variables and the fitness function is the objective function adapted to the search algorithm. The transformation between the model and the representation should be reversible: a solution found in the represented problem should correspond to a solution of the problem and the solution value (fitness value) should be translated to the corresponding objective function value.

It is worth noting that the correspondence between solutions defined in the model solutions and solutions defined in the representation has not to be a one-to-one mapping only. All the three cases are possible: one-to-one (Figure 2a), one-to-many (Figure 2b), many-to-one (Figure 2c). The first case is the most common; for instance, this kind of mapping characterizes problems where variables in the model have finite domains and they are represented just with a different encoding. The second case can be found for example in ACO representations for MAXSAT problems [41], where one assignment corresponds to a set of paths in the construction graph. Finally, a single solution in the "representation space" may correspond to many solutions of the model; for instance, if infeasible solutions are allowed in the representation and they have to be repaired to produce feasible solutions for the model.

[2]This definition of *model* is broader than the usual definition given in Operation Research.

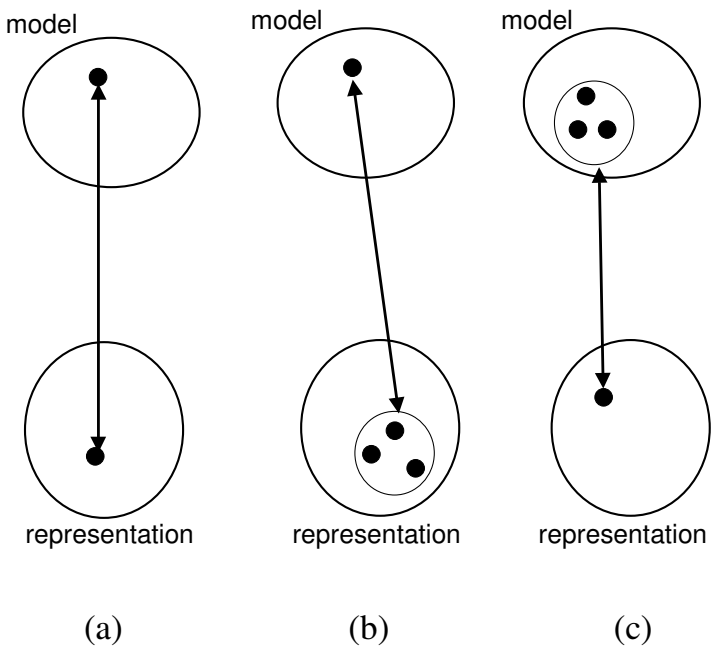

Figure 2. *Three kinds of mapping between model and representation objects. (a) one-to-one, (b) one-to-many, (c) many-to-one.*

To clarify the difference between model and representation, we present two examples. The first example concerns the application of local search to a Constraint Satisfaction Problem; it shows the difference between objectives in the model and the representation. Moreover, it points out that the set of feasible solutions can be different in model and representation. The second example shows a more complex process which leads from the model to a representation suited for a particular metaheuristic (Ant Colony optimization).

3.1.1 Example 1: Local Search for CSP

To solve a CSP one has to find an assignment that satisfies all the constraints of the model. For instance, let us consider a Satisfiability problem: 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 $\{(x_1 \vee \sim x_2), (\sim x_1 \vee \sim x_3), (x_1 \vee x_3)\}$ is satisfied by the assignments $\{(x_1 \leftarrow 1, x_2 \leftarrow 0, x_3 \leftarrow 0), ((x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0))\}$. Since local search can be applied with the notion of objective function, the model of SAT has to be changed into a suited representation, whereby the satisfying assignments have the highest objective function value and the other assignments should have a value function of their "closeness" to the satisfying ones. A typical objective function is evaluated as the number of satisfied clauses, but other choices are possible[3]. To keep distinctions clear, in the following we will call the objective function defined in the representation *fitness function*. This example enlightens also another important difference between model and representation. In a CSP, only the satisfying assignments are feasible solutions, whilst in the representation all the possible assignments are feasible. Hence, in general the set of feasible solutions defined in the model can be different in the representation.

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

3.1.2 Example 2: The Ant Colony Metaheuristic

A more complex example can be found if ACO metaheuristics are applied.

Ant Colony Optimization is a well known metaheuristic for designing algorithms for combinatorial optimization problems [12, 14, 13]. 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 agents that construct a solution by choosing the next component in a probabilistic way (a function of the pheromone). In ACO, pheromone is a mean to communicate good solution components. Several implementations of ACO have been proposed, often with different choices for the construction rules and the way to update pheromone trails [14, 19, 48, 49, 4, 41]. In this section we recall the formal representation of the construction space and the definition of the basic elements of ACO algorithms; for a detailed description see [12].

Artificial ants construct a solution by building a path on a *construction graph* $G = (\mathcal{C}, \mathcal{L})$ and the solution they produce is a sequence of *components*. More formally, the basic elements are:

- A set of components $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_c}\}$
- The *states* of the problem are sequences of components $x = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$. States correspond to partial and complete solutions of the original problem.
- A set of *constraints* which defines the feasible states (i.e., they define the feasible neighborhood of a component).

The graph G is fully connected and its nodes correspond to components; the states are paths in the graph and solutions correspond to the paths completed after the stop condition is met. For example, let us consider the Traveling Salesman Problem:

- nodes of G (the components) are the cities to be visited,
- states are partial or complete paths in the graph,
- a solution is an Hamiltonian tour in the graph,
- constraints are used to avoid cycles (an ant can not visit a city more than once).

The ants behavior is specified by defining *start states* and *termination conditions*, *construction rules*, *pheromone updating rules* and *daemon actions*.

We would like to stress the fact that constraints are first defined in the model, then in the representation and they have different origins and meaning. Furthermore, it is important to observe that the mapping between model and representation has to be taken into account when the metaheuristic algorithm is designed. For example, there is a remarkable difference between ACO application to TSP-like problems and the application for CSPs: in the first case, the order of the components in the path has a fundamental importance, while in the second situation what is important is the final assignment, regardless of the order in which variables have been taken.

3.2 Fitness Landscapes

The search process can be viewed as an exploration in a landscape and the goal is to find a global optimum, or, at least, a "good" local optimum.

A *Fitness Landscape* is defined by a triple:

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

where:

- S is the set of solutions (or states);

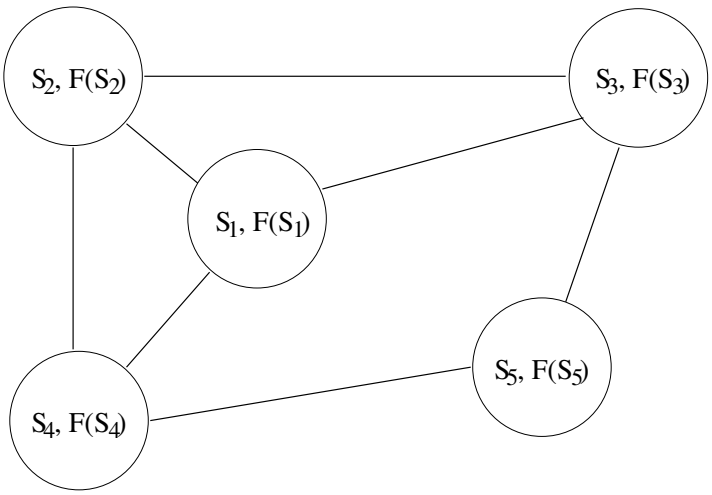

Figure 3. *Example of (undirected) graph representing a fitness landscape. Each node s_i is associated with a solution (state) and its corresponding fitness value $F(s_i)$. Arcs represent transition between states by means of φ (symmetric in this case).*

- $\mathcal{N}$ 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 fitness function, F: $S \to \mathbb{R}$.

The Fitness Landscape (thereinafter referred to as FL) can be interpreted as a graph (see Figure 3) where 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 $s_1 \in S$:

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

Generally, the operator is *symmetric* (i.e., if s_1 is a neighbor of s_2 then s_2 is a neighbor of s_1), but it is not a necessary condition. In a graph representation (like the one depicted in Figure 3) undirected (resp. directed) arcs represent symmetric (resp. asymmetric) neighborhoods.

A property the neighborhood structure should have is to allow a path from every pair of nodes of the graph (i.e, the graph is connected).

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

There exists another definition of fitness landscape, which is more formal and general [28]. 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 substitute a single solution and the operator transforms a multi-set into another. Furthermore, the operator is defined as a function $\psi : M(S) \times M(S) \to [0, 1]$, which assigns a probability for each possible transition between states. This definition of FL allows to deal with population heuristics, like genetic algorithms, in the same way as simple local search algorithms.

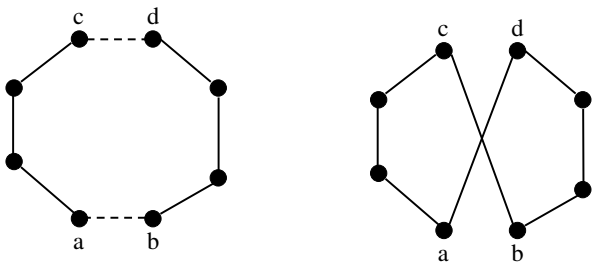

Figure 4. *Example of 2-exchange move. The tour can be reconstructed in only one way.*

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 to a Free Optimization Problem, where there are not constraints and infeasible solutions are penalized by the fitness function [15].

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 [28, 29].

3.2.1 Example: Traveling Salesman Problem

We conclude this section with an example of the process that guide the definition of a representation and subsequently the definition of a FL. The problem we consider is the very well-known Traveling Salesman Problem. The problem can be defined as follows: *given a fully connected undirected graph, with n nodes and each arc is associated with a positive integer value, find the Hamiltonian tour with the minimum total cost.* This definition can be directly translated into a mathematical formalization (the model). Given a model, several representations are possible. We shall discuss two among the most common ones.

Representation 1

A solution (i.e., an Hamiltonian tour) is represented as a sequence of n pairs (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 [27]). The 2-exchange operator deletes two arcs from the tour and reconnects the two paths into a tour in the other way (see Figure 4). In Figure 5 the 3-exchange move is sketched: three arcs are removed and the 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 are different in dimension 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.

Representation 2

A solution (i.e., an Hamiltonian tour) is represented as a sequence of n values $\{v_1, v_2, \ldots, v_n\}$, where v_i are nodes of the graph. 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. Similarly, it is possible to define moves which pick up more than 2 nodes and exchange them, leading to the definition of neighborhoods $\mathcal{N}_{k-perm}$. Observe that these operators

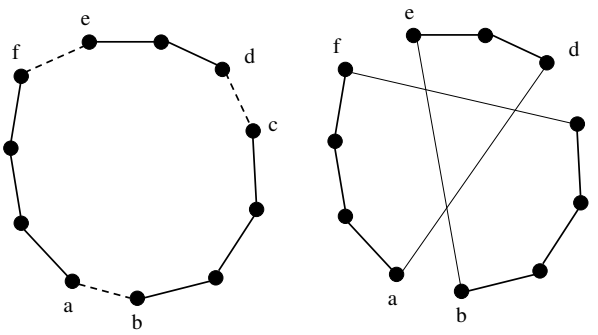

Figure 5. *Example of 3-exchange move. After the removal of three arcs there are six possible ways to reconstruct the tour. In this figure just one of them is shown.*

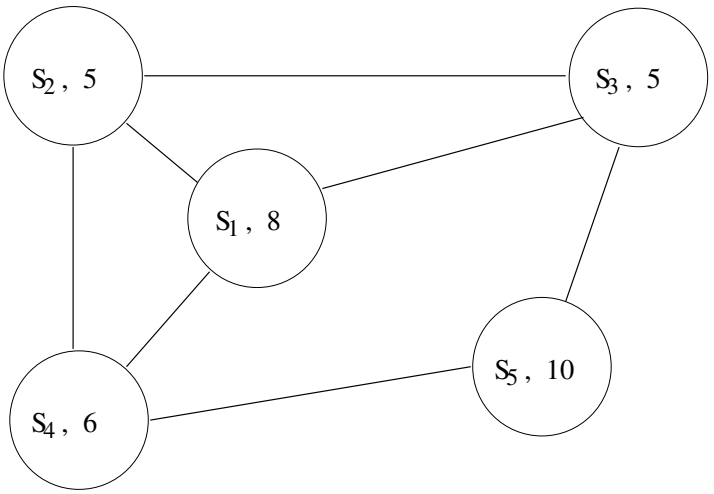

Figure 6. *An example of hill-climbing local search on a FL. If the agent starts at node s_2 it moves toward the best neighbor s_1 whose fitness value is 8. From node s_1 there are no other possibilities to move, because its neighbors have a lower fitness. The agent has reached a local optimum and cannot reach the global optimum s_5.*

define neighborhoods of cardinality proportional to $\frac{n!}{k!(n-k)!}$, thus for large n, $|\mathcal{N}_{k-perm}|$ is larger than $|\mathcal{N}_{h-ex}|$ $(h = 2, 3, \ldots, n)$.

The effectiveness of the representations and FL depends upon the metaheuristic algorithm and the problem structure. In general, no best choice exists which leads to the best performance with every algorithm/problem combination[4].

3.3 Search Agents

The weak notion of *agent* states that an agent is a (software) system that enjoys the following properties [55, 53, 42]:

- *Autonomy*
- *Social ability*
- *Reactivity*

[4]This empirical conjecture is called *No Free Lunch Theorem* and it is theoretically supported by [54].

- *Pro-activeness*

In this context we adopt just the metaphor of agent, referring to a system able to move over a landscape, communicate with other agents, be active (goal-oriented) and, possibly, be adaptive.

Given a FL, an agent implements a search algorithm. It represents a solution (or a set of solutions) during its search. The goal of the agent is to find an optimum of the problem: a global optimum, or a sub-optimal solution[5]. The agent search strategy is described by the algorithm it implements. The environment the agent perceives is the neighborhood of the state (solution) it is in. The information it can access is the set of neighbors and their fitness values.

The simplest case is a hill-climber (best improvement strategy) agent over a FL (Figure 6). The agent starts in a state and it explores the neighborhood, then it chooses the move toward the fittest neighbor. This strategy forces the agent to stop at a local maximum, so there is need to avoid local optima or escaping from them. There are several *smarter* agents, which implement more elaborated search strategies, like Tabu Search (TS), Simulated Annealing (SA), Noising Search (NS) (elsewhere called Local Search plus Random Walk) and so on.

Search strategies which use noise are able to escape from many local optima, since also moves toward less fit states are allowed. The agent metaphor allows to describe the use of *noise* from another perspective. The agent environment perception is, in general, an inner model of the outer reality (see Figure 7). The agent senses the world outside and it tries to model it, or to interpret it. An agent which sees the environment "as it is" enjoys the *perfect information property*. In terms of search agents on FL, this means that it can explore the entire neighborhood and it can calculate the fitness function exact value. Conversely, adding noise to the fitness function means designing an agent with incomplete information about the environment. This implies that it could choose a new state less fit than the present one. A generalized class of noising algorithms can be found in [6]; these algorithms implement in different ways noising strategies and they can be interpreted as basic local search strategies which act in a noised environment. Following this metaphor, even Simulated Annealing and Tabu Search can be interpreted as basic local search agents searching in an environment that is known with a limited precision. In fact both can be interpreted as the action of a simple search agent whose visibility of the environment is limited and noisy. A SA–agent perceives the environment with an increasing precision, since, as time goes on, noise on the evaluation of the fitness decreases. A TS–agent is equipped with a memory which helps it to avoid already visited regions. In practice, a portion of the neighborhood is "obscured" and it is not temporarily visible by the agent. From an algorithmic perspective, this moves the designing issues standpoint from an omniscient state to an individual one. Moreover, it allows to use different agents searching in the same space, but with different perspectives. Now, it has to be pointed out that what in general we call 'search algorithm' (like, for example, TS) is the composition of some mechanisms and it can be interpreted as the interaction/cooperation of agents. In the case of TS, an agent realizes the basic search (the best improvement) in a set of allowed states which is defined by a "strategic" agent that uses tabu and aspiration criteria. It is at this low level of description that we can individuate basic components and adopt the metaphor of noise.

Another more complicated example of the search agent model is the revisitation of memetic algorithms (MAs) [39, 38]. Shortly, MAs are Genetic Algorithms (GAs) with Lamarckian mechanisms. The transmitted genetic information includes also the individual experiences and modifications. Individuals are improved by local search (or a more general trajectory method) and each individual is replaced with the improved solution. This is the equivalent of the experience an individual accumulate during its life. Furthermore,

[5]Without loss of generality, in the following we will consider only maximization problems.

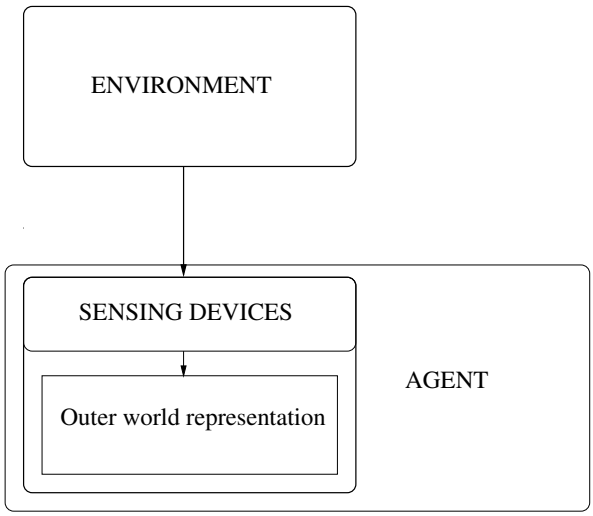

Figure 7. *Agents build a model of the world outside by means of their senses and they use that model to elaborate their strategy.*

random modifications (like mutation) affect every individual and they are transferred to the offspring as well as experience. In metaheuristic applications, MAs can be described as systems composed of several agents. Each agent implements a local search algorithms and it explores its neighboring regions. The fittest agents are those which found the solutions with highest value. Following the genetic algorithms approach, new individuals are generated and the usual operators (mutation, crossover and selection) are applied.

3.4 Cooperative Search

Empirical results (see, for instance, [18]) 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 FL characteristics. Among such properties are: ruggedness, number of local optima, distribution of local optima and topology of the basins of attraction [36, 44, 45, 30, 25, 26]. The choice of fitness function and operator defines such characteristics and it is reasonable that the combination of searches on different FLs (derived from the same model) can overcome the problem hardness (or, at least, effectively face it).

Cooperative search [24, 2, 9, 51, 43] consists of a search performed by some agents which exchange information about states, solutions or other search space characteristics. Agents can be either homogeneous (implementing the search performed by some 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) allow 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* [9].

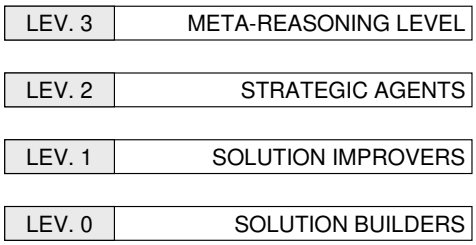

Figure 8. *Multi-level architecture for metaheuristic algorithms.*

The usefulness of the multiagent metaphor emerges when considering design issues like: synchronization, evaluating information, communication, information filtering and implementation. A meaningful example of cooperative search in a multiagent system is reported in [9]: in addition to search agents, there are *referees* agents whose work is to filter information. This architecture allows 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 efficacy and efficiency even if they are sequentially implemented. This means that improvements originate from information exchange and not only from the parallel implementation [35, 31].

4 A multi-level architecture

In the previous section we have reconsidered search algorithms in a multiagent system framework, focusing mainly on trajectory methods. Metaheuristics involve more than simple local search, since they cover all heuristic techniques to solve search problems (solutions construction, improvement and analysis). The multi-level architecture (MLA, depicted in Figure 8) is composed of some levels and each of them corresponds to a different level of *abstraction*. At each level there are specialized agents, each implementing an algorithm. LEVEL–0, provides a feasible solution (or a set of solutions) for the upper level; in this meaning, it is the *feasibility* level. The upper level deals with the improvement of a solution and generally agents perform local search until a local optimum is reached (but there are exceptions, anyway); this can be defined as the level which deals with *local optima*, since the space in which they search is restricted to the neighborhood function. LEVEL–2 should have a global view of the space, or, at least, its task is to govern the search toward promising regions and therefore it can be defined as the *global space* level. The number of levels can be increased by adding another level where learning and reasoning algorithms can be implemented. None of the metaheuristics developed so far needs this level in order to be described, thus the introduction of LEVEL–3 opens the door to the design of new higher level 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 of one specialized for a specific task.

4.1 Definition of the multi-level architecture

LEVEL–0 To this level belong *solutions builder* agents, whose task is to produce starting solutions. Building components are variables or partial solutions, depending on the representation chosen. The builder agents goal is to construct a starting solution for the upper level. Agents may implement algorithms like random initialization, greedy construction, probabilistic construction (e.g. pheromone based), etc. The computational capabilities of LEVEL–0 agents 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). In principle, every algorithm producing a feasible solution can fit the definition of LEVEL–0 agent; therefore even complete algorithms like Branch & Bound could be considered to belong to this level. However, by doing this, we would loose the concept of metaheuristics and, worse, the possibility of describing algorithms as interactions among basic components.

LEVEL–1 This level has been the subject of the previous section. It is the *solution improver* agents level. Agents search in a FL with various algorithms, trying to improve the initial solution they got from the lower level. LEVEL–0 and LEVEL–1 often use the same representation, but they can also use different representations. LEVEL–1 agents implements various search algorithms and they can constitute either single independent search agents, or cooperating agents; they can also realize both trajectory methods and population heuristics. 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, LEVEL–1 agents are usually not just reactive, but they are goal–oriented and they realize a strategy to locally explore the search space.

LEVEL–2 Solutions produced by LEVEL–1 provide very useful information about the structure of optimal or suboptimal solutions. LEVEL–2 agents can memorize the best solutions found, suggest what building blocks are more probably inserted in the optimal solution, what region of the search space is more promising; they can dynamically bias the intensification/diversification ratio, or provide new building blocks for LEVEL–0. LEVEL–2 agents are *building blocks analyzers*: they extract partial solutions which are components of good solutions. They either can be *search space analyzers*: they sample or abstract the search space looking for promising regions to explore. Long term memory is implemented and exploited at this level, where the whole history of the search process is stored. One of the main uses of long term memory is the implementation of diversification strategies. LEVEL–2 agents implement algorithms 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 LEVEL–1 agents.

LEVEL–3 Agents belonging to levels higher than LEVEL–2 coordinate the lower level agents behavior. For example, these 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. At LEVEL–3 agents perform meta–reasoning on the behavior of agents at lower levels. For example, to this level belong agents that filter the information between search agents. The computational capabilities of LEVEL–3 agents can be the highest among all the levels, as they are learning and reasoning agents.

5 MLA and metaheuristic algorithms

MLA 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 MLA.

There are metaheuristics which can just be described by the LEVEL–0 and LEVEL–1 agents (0–agents and 1–agents for brevity); for these algorithms the only requirement is that the agents have a sufficient computational capability to implement the memory structures needed and the conditions that affect the search strategy. For example, the basic Tabu Search can be realized with a 0–agent which randomly constructs an initial solution and a 1–agent which perform the search with one or more tabu lists. Of course,

Algorithm 1 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

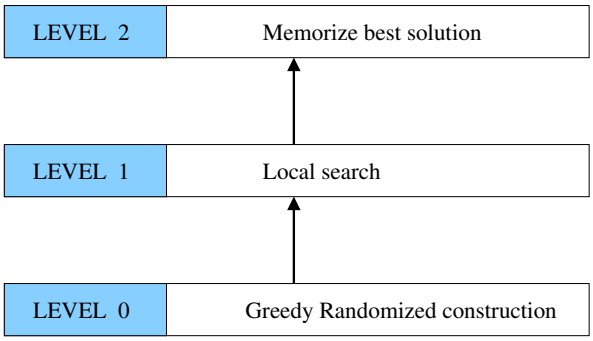

Figure 9. *The Multi-level architecture version of GRASP.*

Tabu Search metaheuristics make often use of long term memory and therefore they need a further agent at LEVEL–2.

The detailed description and discussion of the metaheuristic algorithms we treat is beyond the scope of this paper and they are presented just in their easiest and general form. It is worth noticing 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 and add cooperative search. 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 make use just of some of all the possible communication links.

5.1 GRASP

We start our series of examples with a very simple algorithm: GRASP [16]. This algorithm is composed of a greedy construction procedure and a local search procedure (see Algorithm 1). 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. After this phase, a local search phase takes act; the choice of the local search algorithm depends on the problem to be solved. GRASP is easily described by the MLA: the first level generates a solution by means of a greedy algorithm, 1–agent performs local search, finally 2–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 9 is represented the MLA of GRASP.

5.2 Ant Colony Optimization

Ant Colony Optimization (ACO) has already been introduced in Section 3. In ACO, artificial ants construct a solution by building a path on a *construction graph* $G = (\mathcal{C}, \mathcal{L})$ where the elements of $\mathcal{L}$ (called *connections*) fully connect $\mathcal{C}$ (set of *components*). Artificial

Algorithm 2 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
  Apply local search to every solution in S
  Update pheromone trails
end while

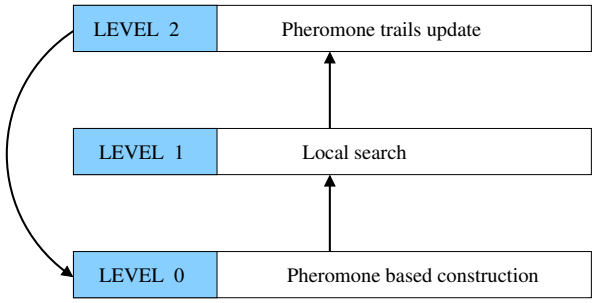

Figure 10. *The Multi-level architecture version of ACO with local search.*

pheromone can be associated either to components (nodes) or connections (edges). The ants behavior is specified by defining *start states* and *termination conditions*, *construction rules*, *pheromone update rules* and *daemon actions*. Often, after the solution construction, ACO algorithms implement a local search phase with the aim to improve the solutions found. A generic ACO algorithm is described in Algorithm 2.

In Figure 10 is depicted the MLA interpretation: 0–agents provide initial solutions by using a constructive procedure biased by pheromone trails. LEVEL–1 agents perform local search and LEVEL–2 agents apply the online delayed pheromone update. The last information (new pheromone values) is then used again by 0–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 is present involving the initial and the final phases.

5.3 Iterated Local Search

Iterated Local Search (ILS) is a simple yet powerful metaheuristic algorithm [47, 46]. 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 which use the history of the search. A very simple version of ILS is described in Algorithm 3.

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

In the MLA framework, ILS can be described as follows (see Figure 11): a 0–agent provides an initial solution (either randomly, or heuristically generated), then it stops its activity. At LEVEL–1 the 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

Algorithm 3 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

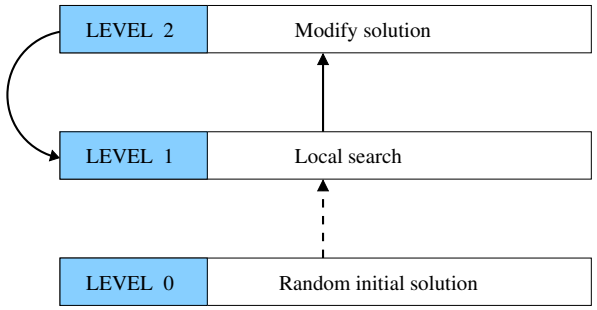

Figure 11. *The Multi-level architecture version of ILS.*

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[6].

5.4 Memetic Algorithms

The last example of the application of the MLA framework is about memetic algorithms (MAs) [39]. We include in this category all the genetic algorithms which apply also local search, in a way similar to ACO plus local search. In a MA (see Algorithm 4), 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 the MLA is described as follows: LEVEL–0 generates an initial population of solutions. This can be done by a single agent which produces the whole population, or a population of agents each one producing a single solution. Furthermore, the initial solution can be generated either randomly, or by means of a construction procedure. 1–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). The task of the LEVEL–2 agents is to combine and/or select solutions and generate a new initial population, to be improved again in LEVEL–1. As we can observe, also in this case the most frequent communication is between LEVEL–1 and LEVEL–2. Figure 12 shows the multi-level architecture for a generic MA.

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

Algorithm 4 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

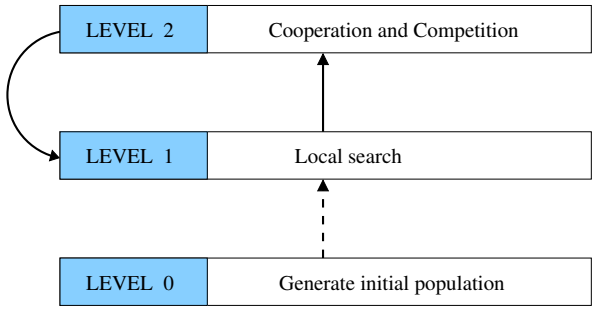

Figure 12. *The Multi-level architecture version of Memetic Algorithms.*

6 Discussion

In the MLA, all direct communications between levels are allowed. In the previous example, 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 allows to clearly individuate information flows. For example, algorithms which have a loop LEVEL–0 $\rightarrow$ LEVEL–1 $\rightarrow$ LEVEL–2 $\rightarrow$ LEVEL–0 exhibit positive and/or negative feedback phenomena. Feedback loops characterize self-adapting systems and emerging phenomena, of which ACO and Evolutionary Computation dynamics are an example.

Another point has to be underlined: different agents can coexist and work concurrently at the same level. They can use different representations but communicate by means of agents which filter and translate information. They realize a kind of *horizontal cooperation*, which has its counterpart in the *vertical cooperation* achieved by the interaction of agents at different levels.

One of the key features of the MLA is the introduction of *meta–levels* (LEVEL–2 and upper levels). 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.). 2–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 realizes the actual search strategy, as it guides the lower level search and adjusts the balance between intensification and diversification.

A new direction the MLA framework opens is *meta–reasoning* performed by LEVEL–3 and LEVEL–2 agents. A way to achieve this is to apply learning techniques (clustering and information extraction, reinforcement learning, etc.). Another very interesting possibility is that 2–agents and 3–agents can be mapped into an optimization problem whose goal is to optimize the quality of information extracted. This can be achieved, for instance, by introducing an entropy measure on the information and the goal is to maximize that value (because the higher the entropy, the higher the quantity of new information). In such a

way, 2 and 3–agents can be realized as search algorithms on LEVEL–1, thus leading to a *meta-search*.

Another relevant consideration is about the synchronization of agents and levels. In the simple examples we have discussed, the easiest way to implement the algorithm is in a sequential and synchronous way: LEVEL–0 produces a solution, then LEVEL–1 improves the solution, finally LEVEL–2 takes some decision about the next neighborhood to explore. But in a MLA architecture, and especially with distributed systems, this choice is restrictive. There is the possibility to implement algorithms in an asynchronous way, providing synchronization mechanisms to the agents and exploiting up–to–date technology for communication and synchronization in multiagent systems. 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 the MLA 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 [33, 37, 10, 11, 17, 1, 8]. 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[7]: construction of the initial solution and further improvement. In [10, 11] an interesting hierarchical structure of object-oriented classes goes into the direction of MLA (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. This taxonomy has been developed to design a framework oriented to the implementation of algorithms and this practical objective is apparent in the presence of the first level of basic software components, which is not considered in our framework. Moreover, the fourth level just represent the outermost cycle of the algorithms and does not realizes the search strategy.

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 [53, 40, 34]. 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 results 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 search algorithms, concepts and researches

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

about the formalization of communication and cooperation among (heterogeneous) agents may be effectively included in the implementation and formalization of the MLA.

8 Conclusion and Future work

In this paper, we have discussed a multiagent approach to metaheuristics. First, we have shown how to view search algorithms as search agents on fitness landscapes. Then, we have described the multi-level architecture, where agents are characterized by different levels of abstraction and problem consciousness. One of the advantages of using the MLA as a framework for metaheuristic algorithms is the easy way of comparing and varying the 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 algorithms implementation. Sequential, parallel (synchronous and asynchronous) implementations can be easily designed. In our opinion the MLA framework for metaheuristics clearly shows that these algorithms are more than simple local optima escaping strategies. They use enhanced "intelligent" strategies to explore the search space, from the learning mechanisms of ACO, to advanced trajectory guidance in ILS.

Future work covers the following topics:

- Studying the use of agents on FL for LEVEL–2 and LEVEL–3 agents behavior.
- Design new hybrid algorithms.
- Explore the possibilities of cooperative search.
- Formalization of the MLA. This work involves the formalization of agents structure and capabilities (by using, for example, a logical language) and the formal specification of communication protocols.
- Use of complex system metaphor; in particular, since the information flow is explicitly traced in the MLA, a characterization of emergence in metaheuristics algorithms can be develop.

Acknowledgments

The authors thank Paolo Torroni for his helpful comments and suggestions on this work.

References

[1] 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.

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

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

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

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

[26] 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.

[27] 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.

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

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

[30] 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.

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

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

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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.

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

[40] 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 Presss/MIT Press, 1996.

[41] 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.

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

[43] 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.

[44] 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.

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

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

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

[48] 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.

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

[50] E.D. Taillard, L.M. Gambardella, M. Gendreau, and J. Potvin. Adaptive memory programming: A unified view of meta-heuristics. Technical report, IDISA, Istituto "Dalle Molle" di Studi sull'Intelligenza Artificiale, Lugano, Switzerland, 1998.

[51] 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.

[52] 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.

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

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

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