TR-H-301

For a Formal Foundation of the Ant Programming Approach to Combinatorial Optimization.

Mauro BIRATTARI (Örebro Univ.),
Gianni DI CARO and Marco DORIGO (IRIDIA, Univ. Libre de Bruxelles)

2000.12.20

ATR人間情報通信研究所
〒619-0288　京都府相楽郡精華町光台2-2-2　TEL: 0774-95-1011

ATR Human Information Processing Research Laboratories
2-2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
Telephone: +81-774-95-1011
Fax : +81-774-95-1008

For a Formal Foundation of the Ant Programming Approach to Combinatorial Optimization

Part 1: The problem, the representation, and the general solution strategy

MAURO BIRATTARI
Learning Systems Lab, AASS
Department of Technology
Örebro Universitet
Örebro, Sweden
mbiro@aass.oru.se

GIANNI DI CARO
Department of Evolutionary Systems
Human Information Processing Labs
Advanced Telecommunications Research (ATR)
Kyoto, Japan
d_ikarus@yahoo.com

MARCO DORIGO
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium
mborigo@ulb.ac.be

Abstract

This paper develops the formal framework of *ant programming* with the goal of gaining a deeper understanding on *ant colony optimization*, a heuristic method for combinatorial optimization problems inspired by the foraging behavior of ants. Indeed, *ant programming* allows a deeper insight into the general principles underlying the use of an iterated Monte Carlo approach for the multi-stage solution of a combinatorial optimization problem. Such an insight is intended to provide the designer of algorithms with new categories, an expressive terminology, and tools for dealing effectively with the peculiarities of the problem at hand.

Ant programming searches for the optimal policy of a multi-stage decision problem to which the original combinatorial problem is reduced.

In order to describe *ant programming*, the paper adopts on the one hand concepts of optimal control, and on the other hand the *ant* metaphor suggested by *ant colony optimization*. In this context, a critical analysis is given of notions as state, representation, and sequential decision process under incomplete information.

1 Introduction

In the last decade, starting from the work of Dorigo and co-workers [13, 7, 14, 12] in 1991, a number of algorithms inspired by the foraging behavior of ant colonies has been introduced for solving combinatorial optimization problems heuristically (see [10, 9, 11] for extensive reviews). With the aim of giving a first unifying description of (most of) such algorithms, the framework of *ant colony optimization* has been recently proposed [9, 10].

According to such a framework, with *ant colony optimization* we will refer in this paper to any algorithm that iteratively generates and evaluates paths on a weighted graph which conveniently encodes the combinatorial problem at hand. The graph is such that each solution of the combinatorial problem can be put in correspondence to a path on the graph. The weights associated to the edges of the graph are such that the cost of a path, i.e. the sum of the weights of its composing edges, equals the cost function of the combinatorial problem for the associated solution. In this sense, the goal of *ant colony optimization* is to find a path of minimum cost. To this end, a number of paths are generated in a Monte Carlo fashion, and the cost of such paths is used to bias the generation of further paths. This process is iterated with the aim of gathering more and more information on the graph to eventually produce a path of minimum cost.

In *ant colony optimization*, the above described algorithm is visualized in terms of a metaphor in which the generation of a path is described as the walk of an *ant* that, at each node, stochastically selects the following one on the basis of local information called *pheromone trail* [1]. In turn, the *pheromone trail* is modified by the *ants* themselves in order to bias the generation of future paths toward better solutions.

Ant colony optimization has been successfully applied to a number of different problems.[1] Nevertheless, a complete theoretical analysis of the algorithms of this class is not available yet.

The aim of this paper is to produce a formal description of the combinatorial optimization problems to which *ant colony optimization* applies, and to analyze the implication of adopting a generic solution strategy based on the incremental Monte Carlo construction of solutions. In particular, the paper proposes a critical analysis of the concept of *state* in the context of a sequential decision process. Further, it introduces *ant programming* as an abstract class of algorithms which presents all the characterizing features of *ant colony optimization*, but which is more amenable to theoretical analysis. In particular, *ant programming* bridges the terminological gap between *ant colony optimization* and the fields of optimal

[1] The interested reader can found in [9, 11, 10] lists and descriptions of implementations of *ant colony optimization* for a variety of combinatorial optimization problems like, among others, traveling salesman, quadratic assignment, and graph coloring. Additionally, overviews of algorithms inspired by the behavior of real ants but not strictly falling in the *ant colony optimization* class, are reported in [5, 6, 8].

control and reinforcement learning, with the final goal of exploiting the understanding gained in those fields for the analysis of *ant colony optimization.*

The name *ant programming* has been chosen for its assonance with *dynamic programming*, with which *ant programming* has in common the idea of reformulating an optimization problem as a multi-stage decision problem. Both in dynamic programming and in *ant programming*, such a reformulation is not a trivial one and requires an *ad hoc* analysis of the optimization problem under consideration. Once the multi-stage decision problem is defined, in *ant programming* as in dynamic programming, the optimal solution of the original optimization problem can be generated through the optimal *policy* of the multi-stage decision problem. These concepts, being among the main issues in this research, will be discussed in detail in the rest of the paper.

In Section 2 we show how the generic problem of discrete optimization with a finite number of solutions can be described in terms of a discrete-time optimal control problem which, in turn, can be mapped on the problem of searching the path of minimal cost on a conveniently defined weighted graph.

Section 3 introduces the concepts of *graph of the representation* and of *phantasma.* These concepts, which are related to the notion of *sequential decision process under incomplete information*, will play a major role in the definition of *ant programming.*

In Section 4 the *ant programming* abstract class of algorithms is introduced and discussed. Moreover, a well defined semantic is given to all the elements of the *ant metaphor.*

Section 5 describes the future developments of our research, and comments on the significance of *ant programming* as a basis for providing *ant colony optimization* with a new formal definition and a theoretical explanation. In particular, we briefly introduce here two instances of *ant programming*, namely: *Markov ants* and *Marco's ants.* On the one hand, as it will be made clear in the body of the paper, the *Markov ants* algorithm is mainly of speculative interest, and is to be considered as the *extreme* end of the *ant programming* class when the above mentioned sequential decision process is carried out under *complete* information. On the other hand, the *Marco's ants* algorithm is of much greater practical interest and happens to present the characterizing features of the implementations of *ant colony optimization* proposed so far in the literature, starting from the original *ant system* introduced by Marco Dorigo *et al.*, back in 1991 [13, 7].

2 Discrete optimization, optimal control, and paths on a graph

Let us consider a discrete optimization problem described by a finite set S of feasible solutions, and by a cost function J defined on such a set.

The set S of feasible solutions is defined as

$$S = \{s_1, s_2, \ldots, s_N\}, \quad N \in \mathbb{N}, \quad N < \infty, \tag{1}$$

where each solution s_i is a n_i-tuple

$$s_i = (s_i^1, s_i^2, \ldots, s_i^{n_i}), \quad n_i \in \mathbb{N}, \quad n_i \leq n < \infty, \tag{2}$$

with $n = \max n_i$, and $s_i^j \in Y$, where Y is a finite set of *components*.

The cost function $J : S \to \mathbb{R}$ assigns a cost to each feasible solution s_i. The optimization problem is therefore the problem of finding the element $\bar{s} \in S$ which minimizes the function J:

$$\bar{s} = \arg\min_{s \in S} J(s), \tag{3}$$

where with "arg min" we denote the element of the set S for which the minimum is attained.[2]

2.1 Incremental construction of a solution for a discrete optimization problem

A feasible solution can be obtained incrementally starting from the 0-tuple, hereafter $x_0 = ()$, and adding one-at-a-time a component. The generic iteration of this process can be described with the following notation:

$$x_j = (u_0, \ldots, u_{j-1}) \to x_{j+1} = (u_0, \ldots, u_{j-1}, u_j), \quad \text{with } u_j \in Y, \tag{4}$$

where x_j will be called a *partial solution* of length j. In order to guarantee that the solution being built be feasible, we require that the $(j+1)$th component considered be such that it exists at least one feasible solution $s_i \in S$ of which x_{j+1} is an "initial sub-tuple" of length $j+1$. More formally:

$x_{j+1} = (u_0, \ldots, u_j)$ is feasible if and only if

$$\exists s_i : \ s_i \in S \text{ and } s_i^k = u_{k-1}, \text{ with } 1 \leq k \leq j+1 \leq n_i. \tag{5}$$

If the partial solution x_j is feasible, in order to guarantee the feasibility of x_{j+1} it is sufficient to check the constraint (5) only for $k = j+1$. We can therefore define the set $U(x_j) \in Y$ of all the possible new components u_j that can be appended to x_j giving in turn a feasible solution x_{j+1}:

$$U(x_j) = \left\{ u_j \mid \exists s_i : \ s_i \in S \text{ and } s_i^{j+1} = u_j \right\}. \tag{6}$$

From the fact that the sets S and Y are finite and that the length of each feasible solution s_i is finite as well, it follows that it is finite also the set X of all the feasible tuples x_j, i.e. the set of all the tuples built according to the rule (4) and satisfying the constraint (5). Moreover, it can be shown that $S \subset X$, since all the solutions s_i are composed by a finite number of components all belonging to Y.

[2] Being the set S finite, the minimum of J on S indeed exists. If such minimum is attained for more than one element of S, it is a matter of indifference which one is considered.

2.2 Multistage decision processes and optimal control

The incremental construction of a feasible solution illustrated in (4) can be used in order to solve the original optimization problem through a *multi-stage decision process* in which the optimal solution $\bar{s}$ is obtained by a sequence of decisions concerning the set Y of the components. In other words, instead of solving the problem (3) directly on the set S of the solutions, we might solve a sequence of sub-problems, the $(j+1)$th of which consisting in selecting a component $u_j \in Y$ to be appended to the partial solution x_j in order to obtain the (partial) solution x_{j+1}.

Such a way to proceed results particularly natural when the cost $J(s_i)$ of a solution s_i is expressed as a sum of contributions c_{j+1}, each related to the fact that a particular component u_j is included in the solution s_i itself after a sequence of components described by the tuple x_j.

More formally, this means that a function $C : X \setminus \{x_0\} \to \mathbb{R}$ must be conveniently defined, which associates a cost c_{j+1} to each tuple x_{j+1}, where x_{j+1} is obtained by appending the new component u_j to the current partial solution x_j.[3] The function C must be such that

$$J(s_i) = \sum_{j=1}^{n_i} c_j, \quad \text{with } c_j = C(x_j), \text{ and where } x_j = (u_0, \ldots, u_{j-1}), \text{ with } u_{k-1} = s_i^k \text{ for } 1 \leq k \leq j. \quad (7)$$

Although a function C satisfying (7) can always be trivially obtained by imposing $C(x_j) = J(s_i)$, if $\exists s_i \in S : x_j = s_i$, and $C(x_j) = 0$ otherwise, we suppose here that a non-trivial definition exists, and that its formulation is somewhat "natural" for the problem at hand.

The multi-stage decision process described above can be thoroughly seen as a *discrete-time optimal control problem* [4].

The tuple x_j can be seen as the *state* at time $t = j$ of the following discrete-time dynamic system:

$$\begin{cases} x_{t+1} = F(x_t, u_t), \\ y_{t+1} = G(x_{t+1}), \end{cases} \quad (8)$$

with $t \in \mathbb{N}$, $x_t, x_{t+1} \in X$, where X is the set of the states of the system, $y_{t+1} \in Y$, where Y is the range of the output, and $u_t \in U(x_t)$ where $U(x_t)$ is the set of the *control actions* which are admissible when the system is in state x_t.

[3] It is worth noticing here that, given rule (4), the tuple x_{j+1} determines univocally the tuple x_j and the component u_j, and is in turn determined univocally by them. As a result, the function C could be equivalently defined as a function mapping on the real line an ordered pair $\langle x_j, u_j \rangle$, a transition $\langle x_j, x_{j+1} \rangle$, or even the triplet $\langle x_j, u_j, x_{j+1} \rangle$.

Such a process is always started from the same initial state x_0, and is terminated when the state belongs to the set $S \subset X$, defined as the set of the final states of the control process.

The state-transition application $F : X \times U \to X$ is such that the state at time $t+1$ is obtained by "appending" the current control action $u_t \in U(x_t)$ to the state x_t. Further, the function $G : X \to Y$ is such that the new component is observed as the output at time $t+1$ of the system itself. Therefore, the following notation will be adopted:

$$\begin{cases} x_{t+1} = [x_t, u_t], \\ y_{t+1} = u_t, \end{cases} \tag{9}$$

It results therefore that the set of the feasible actions, given the current state, is a subset of the range of the output: $U(x_t) \subset Y$.

Let now $\mathcal{U}$ be the set of all the admissible control sequences which bring the system from the initial state x_0 to a terminal state. The generic element of $\mathcal{U}$,

$$u = \langle u_0, u_1, \ldots, u_{\tau-1} \rangle,$$

is such that the corresponding state trajectory, which is unique, is

$$\langle x_0, x_1, \ldots, x_\tau \rangle,$$

with $x_\tau \in S$, and $u_t \in U(x_t)$, for $0 \leq t < \tau$. In this sense, the dynamic system defines a mapping $\mathcal{S} : \mathcal{U} \to S$ which assigns to each admissible control sequence $u \in \mathcal{U}$ a final state $s = \mathcal{S}(u) \in S$.

The problem of optimal control consists in finding the sequence $\bar{u} \in \mathcal{U}$ for which the sum J of the costs c_t, incurred along the state trajectory, is minimized:

$$\bar{u} = \arg\min_{u \in \mathcal{U}} J\big(\mathcal{S}(u)\big), \tag{10}$$

where with "arg min" we denote the element of the set $\mathcal{U}$ for which the minimum of the composed function $J \circ S$ is attained.[4]

2.3 Paths on a graph

It is apparent that the solution of the problem of optimal control stated in (10) is equivalent to the solution of the original optimization problem (3), and that the optimal sequence of control actions $\bar{u}$ for the optimal control problem determines univocally the optimal solution $\bar{s}$ of the original optimization problem.

Since the set X is discrete and finite, together with all the sets $U(x_t)$, for all $x_t \in X$, and since the state-transitions are governed by the application F and are

[4]The same remarks of Note 2 apply here. In particular, since the set $\mathcal{U}$ is finite, it is guaranteed that $J \circ S$ attains its minimum on $\mathcal{U}$.

therefore deterministic, all the possible state trajectories of the system (8) can be conveniently represented through a weighted and oriented graph with a finite number of nodes.[5]

Let $\mathcal{G}(X, U, C)$ be such a graph, where X is the set of the nodes, U is the set of the edges, and C is a function that associates a weight to each edge. In terms of the system (8), the notation adopted means that each node of the graph $\mathcal{G}(X, U, C)$ represents a state x_t of the system. The set $U \subset X \times X$ is the set of the edges $\langle x_t, x_{t+1} \rangle$. Each of the edges departing from a given node x_t represents one of the actions $u_t \in U(x_t)$ feasible when the system is in state x_t. Therefore,

$$U = \bigcup_{x_t \in X} U(x_t).$$

Finally, $C : U \to \mathbb{R}$ is a function that associates a cost to every edge. Such a function is defined in terms of the function $\mathcal{C}$, and it results that

$$c_{t+1} = C(\langle x_t, x_{t+1} \rangle) = \mathcal{C}(x_{t+1})$$

is the cost of the edge $\langle x_t, x_{t+1} \rangle$.

Furthermore, on such a graph $\mathcal{G}(X, U, C)$ it can be singled out the initial state x_0, as the only state with no incoming edges, and the set S of the terminal nodes from which no edges depart.

In terms of the graph $\mathcal{G}(X, U, C)$ the optimal control problem (10) can be stated as the problem of finding the *path of minimal cost* from the initial node x_0 to any of the terminal nodes in S.

It is worth noticing here that the set of nodes X can be partitioned in $n + 1$ subsets:

$$X = X_0 \cup X_1 \cup \cdots \cup X_n, \quad \text{with } X_i \cap X_j = \emptyset, \text{ for } i \neq j,$$

where n is the length of the longest solution in S. The generic subset X_i contains all and only the nodes x_i such that the state they represent is a tuple of i components.

As a consequence of the definition of the state-transition application F, the edges departing from a non-terminal node $x_i \in X_i$, enter a node $x_{i+1} \in X_{i+1}$. The graph $\mathcal{G}(X, U, C)$ is therefore "sequential" and accordingly enjoys the property, remarkable in this context, of being acyclic.

As already mentioned in Section 1, the solution strategy of *ant colony optimization* is based on the iterated generation of multiple paths on a graph that

[5]Remark on the notation adopted: Parenthesis $(\dots)$ are used for tuples of components indicating the state of the system (8) or more in general a (partial) solution of the optimization problem (3). Angle brackets $\langle \dots \rangle$ are used for generic tuples: e.g., an edge of a graph is represented as a *pair* $\langle x_t, x_{t+1} \rangle$ of nodes. Square brackets $[\cdot, \cdot]$ indicates the operation of appending an element to a tuple: if x_t is a tuple of length t, $x_{t+1} = [x_t, u_t]$ is a tuple of length $t + 1$ obtained by appending u_t to x_t. Braces $\{\dots\}$ are used, as usual, for denoting a set.

encodes the optimization problem under consideration. As it will be defined in the following, this graph is obtained as a "transformation" of the graph $\mathcal{G}$. In previous works on *ant colony optimization*, the graph resulting from such transformation was the only graph taken into consideration explicitly. In the present paper, we move the focus on the original graph $\mathcal{G}$ and on the properties of the transformation.

The importance of the graph $\mathcal{G}$ is related to the fact that it brings all the information relevant for a multi-stage solution of the optimization task, as it is carried out in *ant colony optimization*. More in general, the concepts and the terminology peculiar to control theory and to the multi-stage solution of the original optimization problem (3), holds fundamental importance in our analysis of *ant colony optimization*, and constitutes the starting point for our discussion on *ant programming*. In particular, in the following we will adopt the language and the notation of control theory and we will focus on the analysis of concepts like those of *state* and *policy* which are clearly defined in optimal control and dynamic programming.

Before going to the core of these topics in the next sections, it is worth to give a more defined characterization of the class of "real" problems that *ant colony optimization* is meant to deal with. As previously sketched, these algorithms look for minimum cost paths on a graph. Being minimum cost path problems of great practical and theoretical interest, a large number of heuristic or exact algorithms has been proposed for their solution. The approach proposed in this paper is not to be considered as a general alternative to algorithms based on the "classic" label setting techniques (e.g. Dijkstra algorithm) and label correcting techniques (e.g. Bellman-Ford algorithm). On the contrary, it is intended to be used when the dimension of the graph $\mathcal{G}$ overwhelm the available computational resources as, for example, in large instances of NP-hard problems.

3 Markov and non-Markov representations: The state and its *phantasma*

According to the parlance of optimal control, we have called a *state* each node of the graph $\mathcal{G}(X, U, C)$ and, by extension, we use the expression *state graph* for the graph $\mathcal{G}$ itself. In the following sections, the properties of the state graph will be analyzed and discussed in the perspective of the solution of the problem (3), and in relation to the solution strategy of *ant colony optimization*. The ant metaphor introduced by *ant colony optimization* will be used as a visual tool to make abstract concepts more clear. In particular, we will picture the state evolution of the system (9), and therefore the incremental construction of a solution as the walk of an *ant* on the state graph $\mathcal{G}$. Accordingly, in the following the state x_t at time t will be called interchangeably the "partial solution," the "state of the

system," or, by extension, the "state of the ant."

3.1 The concept of state in control theory

Informally, the state of a dynamic system can be thought of as the piece of information that "completely" describes the system at a given time instant.

In more precise terms, for a deterministic discrete-time dynamic system, as the one introduced in Eq. 8, the state is a *tag* or *label* that can be associated to a particular aggregate of input-output histories of a system, and that enjoys the following properties:

- At a given instant, the set of the admissible input sequences can be given in terms of the state at that instant;
- For all the admissible future input sequences, the state and a given admissible input sequence determine univocally the future output trajectory.

It is very important that the reader be convinced of the relevance of both the above stated properties. The following argumentation might help clarifying. Let us say that, after being fed with a certain input sequence and having returned a corresponding output sequence, the system at hand is left in a certain *condition.* Given that the system is deterministic, for any possible continuation of the input, the output will be univocally determined. Now, we say that two given conditions are *equivalent* and are to be described by one and the same *state*, if they are indistinguishable to a class of experiments defined as follows. We consider the experiments that consist in feeding the system in the two different conditions with the same input sequence, and we will say that the two conditions are indistinguishable to the given input sequence if the two corresponding output sequences are equal. Therefore, the two conditions are to be labelled with the same state if and only if they react with the same output sequence to *any* possible input sequence. The two above stated properties follow directly. In particular, we want to stress here the importance of the first one: If there exists an input sequence that is admissible for one of the two conditions and not for the other, such an input sequence is an experiment that enables us to distinguish the two conditions that therefore cannot be labeled with the same state. Hence it is clear that, in order for the two conditions to be equivalents and to be described with one and the same state, it is necessary that they have the same set of admissible inputs. In other words, all the conditions represented by the same state share the same admissible inputs. Therefore the set of the admissible inputs is a property of each state, i.e. such a set can be expressed as a function of the state, as indeed stated by the first of the two properties.

The definition of state given here is inspired by and is consistent with the classical definition given by Zadeh and Desoer [20]. It is nonetheless a proper extension of that definition in the sense that we consider here systems for which

the set of admissible inputs varies according to the preceding input-output history, while Zadeh and Desoer considered systems that always accept input in the same given range. In this sense, our definition of state directly corresponds to the one implicitly assumed in the literature on dynamic programming [2, 3].

Even if this paper focuses on deterministic systems, another class of dynamic systems, known as *stochastic dynamic systems*, is of particular interest. In a stochastic dynamic system, the output at time $t+1$ cannot be expressed deterministically as a function of the state and of the input at time t and, at most, some probability distribution can be given on its possible values. In general, a stochastic dynamic system can be obtained from a deterministic one if some state components of the deterministic system are not available or are purposely not taken into consideration.

Also in the case of a stochastic dynamic system, a definition of a state can be given. In the stochastic case, instead of determining univocally the future output trajectory, the state determines the probabilistic distribution of the future output, and is such that past values of the states and of the input would not change the probabilistic distribution if used to extend the state itself.

In more precise terms, x_t is the state of the stochastic system at time t if it determines the set $U(x_t)$ of the admissible control, and

$$P_k(y_{t+k}|u, x_t) = P_k(y_{t+k}|u, x_t, u_{t-1}, x_{t-1}, \ldots, u_{t-m}, x_{t-m}),$$
$$\text{for all } k > 0,\ m > 0,$$
$$\text{and for all admissible } u = \langle u_t, \ldots, u_{t+k-1} \rangle. \quad (11)$$

By extension, a law of evolution of the state can be given only stochastically, and the state at time $t+1$ is a random variable whose value is extracted according to the distribution $P(x_{t+1}|u_t, x_t)$ where again

$$P(x_{t+1}|u_t, x_t) = P(x_{t+1}|u_t, x_t, u_{t-1}, x_{t-1}, \ldots, u_{t-m}, x_{t-m}),$$
$$\text{for all } m > 0, \text{ and for all } u_t \in U(x_t). \quad (12)$$

For the class of systems considered in this paper, it is worth noticing that the definition of state given above for the stochastic system, indeed encompasses the one given for the deterministic system. In fact, in order to consider a deterministic system, it is sufficient to define the distribution (12) so that the probability of x_{t+1} conditioned to x_t and u_t, equals 1 if and only if, according to the deterministic system (8), x_{t+1} follows x_t when the input u_t is applied.

As a consequence, both in the deterministic and in the stochastic case, we can informally think of the state as the "most predictive" description that can be given of the system.

3.2 Markov property and the concept of representation

Since what is known in the literature as *Markov property* is related precisely to the above stated definition of state, it is clear that the state, when correctly conceived, is *always* a state in the Markov sense: When described in terms of its state, any discrete-time system[6] is *intrinsically* Markov in the precise sense given above. It is therefore of dubious utility to state the Markov property with respect to a dynamic system *tout court*.

Of much greater significance, it is to assert the Markov property of a *representation*. Informally, we call a representation the information on the system that is available to an agent and that is used in order to perform a prediction or to select a control action.

In the limit, a representation might bear the same information as the state. In this case we speak of a Markov representation, or, equivalently we say that the Markov property holds for such representation. These expressions are to be understood as an assertion of the fact that the representation considered is equivalent to a state description of the system. In the more general case, a representation gives "less information" than the state. Consistently, in this case we say that the representation is of non-Markov type.

Being non-Markov is therefore a characteristic of the interaction system-agent and is related to the fact that the agent describes the system in terms of an "incomplete" representation. In general, such a shortcoming of the representation can be ascribed to the inability of the agent to obtain information on the system, or to the deliberate choice of reducing the amount of information to be handled. In this second case, we can speak of a trade-off between the "quality" of a description, and its "complexity."

3.3 Generating a representation: from the state to the *phantasma*

For the class of problems discussed in this paper and generally described by Eq. 3, a formal definition of a representation can be given, which refers directly to the definition given above of the state graph $\mathcal{G}(X, U, C)$. Consistently with that definition, we define the *representation graph* as the graph $\mathcal{G}_r(Z_r, U_r, T)$, where Z_r is the set of the nodes, and U_r is the set of the edges. The meaning and usage of the function $T : U_r \to \mathbb{R}$ will be made clear in the following.

Furthermore, we call *generating function of the representation* the function $r : X \to Z_r$ that maps the set X of the states *onto* the set Z_r. The function r associates therefore to every elements of X an element in Z_r, moreover every element $z_t \in Z_r$ has *at least* one preimage in X, but the preimage needs not be

[6] What follows is true for a larger class of systems. Anyway, for definiteness, we restrict the focus on the subclass which is of interest in this paper.

unique. We adopt the notation

$$r^{-1}(\{z_t\}) = \{x_\tau | r(x_\tau) = z_t\}$$

to describe the set of states x_τ whose image under r is z_t. The function r induces an equivalence relation on X: Two states x_i and x_j are *equivalent* according to the representation defined by r, if and only if $r(x_i) = r(x_j)$. In this sense, a representation can be seen as a *partition* of the set X of the states.

In the following, we will call each $z_t \in Z_\tau$ a *phantasma*, adopting the term used by Aristotle with the meaning of *mental image*,[7] and reintroduced in Medieval epistemology by Thomas Aquinas.[8]

By using such a term we want to stress that, from the point of view of an agent that observes the system through the representation r, z_t plays the role of the *phenomenal perception* of the system itself, that is, all what is known and retained about the system at time t. The word *phantasma*, in the above mentioned meaning, can be considered as a synonym of *image*, term that bears an assonance with the mathematical concept of image: The *phantasma* $z_t = r(x_t)$ is indeed what in elementary mathematics is called the image of x_t under the mapping r.

Thanks to the notion of *phantasma*, we can give now a precise interpretation to the concept of representation. As we pointed out before, the state evolution of the system (8) can be pictorially described as the walk of an *ant* on the graph $\mathcal{G}(X, U, C)$. Let us assume now that the *ant* visits in sequence the nodes $x = \langle x_0, x_1, \ldots, x_n \rangle$. The same sequence, under the lens of the representation generated by the function r, appears to the *ant* as a sequence $z = \langle z_0, z_1, \ldots, z_n \rangle$ where for each i, with $0 \leq i \leq n$, z_i is the *phantasma* of the state x_i, that is: $z_i = r(x_i)$. Being the state graph $\mathcal{G}(X, U, C)$ acyclic, the terms of the sequence x are distinct. Nonetheless, since the function r is not generally one-to-one, the sequence z might contain the same element more than once. It follows therefore that the graph $\mathcal{G}_r(Z_r, U_r, T)$ in the general case may present cycles. Sticking to the *ant* metaphor, we could say that the *ant*, though moving on the state graph $\mathcal{G}(X, U, C)$, *represents* its movement on the nodes $z_i \in Z_r$ of the representation graph $\mathcal{G}_r(Z_r, U_r, T)$.

In the literature on control theory and system identification, the above described process which carries the state into what we call here *phantasma*, is related to the concept of *state-space reduction*.[9] Another notion that arises in

[7] Aristotle (384–322 BC) *De Anima*: "The soul never thinks without a mental image *(phantasma)*."

[8] Thomas Aquinas (1225–1274) *Summa Theologiae*.

[9] Nevertheless, the result of a *state-space reduction* does not have a standard name in system and control theory. Moreover, the terms used in the literature always bring a direct reference to the concept of state: i.e. *reduced state*. It is just in order to underline the important qualitative difference between the properties of the state and those of the result of a *state-space reduction*, that we introduce here the term *phantasma* to denote the latter.

control theory and system identification, and that is significantly related to the concept of *phantasma*, is the notion of (incomplete) *state reconstruction*. Both concepts [15] are related to the necessity of gathering information on the state of the system in order to perform a control action, nonetheless, it is important to keep these two concepts separated.

The problem of state reconstruction arises when it is not possible to have access immediately to the state of the system, usually a physical entity, but related quantities are readable and have been read and stored in some appropriate preceding time window. The case discussed in this paper is significantly different. The system (8) is a mathematical entity that bears no direct relation with any physical device: It is simply an abstract description of the incremental construction of a feasible solution of the optimization problem (3). The state of such a system is therefore immediately readable and, in principle, usable in order to select a feasible control action. Still, it is easy to verify that in many cases of great practical importance, think for example of NP-hard problems, the number of nodes of the state graph grows more than polynomially with the number of problem components, making infeasible any solution method that relies directly on a state description. Therefore, it might become desirable, or even necessary, the use of some quantity related to the state but more easily manageable. This quantity, that can be obtained via a heuristic method that allows to move from the state graph $\mathcal{G}$ of the system to a smaller and more manageable representation graph $\mathcal{G}_r$, is the *phantasma* discussed in this paper.

This change of representation is by no means free from complications. While in the graph $\mathcal{G}$ every (partial) path is a (partial) feasible solution and *vice versa*, in the graph $\mathcal{G}_r$ this property does not hold anymore: to every (partial) solution a path on the graph $\mathcal{G}_r$ can be associated, but the opposite in not true. As far as the construction of feasible solutions is concerned, the graph $\mathcal{G}$ is not therefore superseded by $\mathcal{G}_r$.

As anticipated before, the graph $\mathcal{G}_r$ and the information stored on it are used to take decisions while a solution is being built. Because of the loss of topological information induced by the transformation from $\mathcal{G}$ to $\mathcal{G}_r$, in the general case only sub-optimal solutions will be obtained.

In this sense, while the reconstruction of the state is related to a process in which one indeed tries to *construct* a quantity related to the state starting from simpler pieces of information that are directly readable, a *phantasma* is related to a reduction process that goes in the opposite direction: The state is itself directly available but it is *de-constructed* in order to obtain a representation that be more manageable.

In the same spirit of the definition of the set Z_r, also the set of the edges U_r can be defined in terms of the generating function r. The set $U_r \subset Z_r \times Z_r$ is the set of the edges $\langle z_i, z_j \rangle$ for which it exists an edge of the state graph $\langle x_i, x_j \rangle \in U$,

such that x_i and x_j are the preimages under r of z_i and z_j, respectively. Formally:

$$U_r = \Big\{ \langle z_i, z_j \rangle \mid \exists \langle x_i, x_j \rangle \in U : z_i = r(x_i),\ z_j = r(x_j) \Big\}.$$

The major difficulty that arises when the system is described through a generic representation r, is that the subset $U_r(t) \subset U_r$ of the admissible control actions at time t, usually cannot be described in terms of the *phantasma* z_t alone, but needs for its definition the knowledge of the underlying state x_t. More in detail, when the agent observes the *phantasma* z_t, a transition $\langle z_t, z_{t+1} \rangle$ towards the *phantasma* z_{t+1} is admissible only if it exists the edge $\langle x_t, x_{t+1} \rangle \in U(x_t)$, where x_t, among all the preimages $r^{-1}(\{z_t\})$ of the *phantasma* z_t, is the *actual* current state which gave rise to z_t itself, and where $r(x_{t+1}) = z_{t+1}$.

Such a fact shows clearly that for the generic generating function r, the *phantasma* does not enjoy the first property of the definition of state given above and therefore the corresponding representation is non-Markov. Anyway, it is always possible to obtain *at least* one representation that enjoys the Markov property. In particular, in order to obtain a Markov representation it is sufficient that the function r be one-to-one, and the most trivial Markov representation can be obtained by posing $r \equiv I$, where I is the identity function.

In the general case, the parallel of the function C of $\mathcal{G}$ for the graph $\mathcal{G}_r$ cannot be defined in a straightforward manner. Moreover, it results more useful to define the weights of the edges of the graph $\mathcal{G}_r(Z_r, U_r, T)$ so that they describe the quantity that in *ant colony optimization* is called *pheromone trail*. The function $T : U_r \to \mathbb{R}$ will be used in the process of selecting an action by an *ant* when perceiving a given *phantasma*, and will be iteratively modified in order to improve the quality of the solutions generated. A thorough definition and analysis of the function T will be given in Section 4.

It is anyway worth discussing here the difficulties that arise when trying to define, on the basis of C, a function that maps an edge $\langle z_t, z_{t+1} \rangle$ of $\mathcal{G}_r$ to a cost. Let the agent that observes the system through the representation r see, for instance, a transition from the *phantasma* z_t to the *phantasma* z_{t+1}, without being aware of the details of the underlying state transition from x_t to x_{t+1}. The only thing that the agent is able to induce about the state description from his observations, is that some transition has occurred between a state belonging to the set of preimages $r^{-1}(\{z_t\})$ of the *phantasma* z_t, to one belonging to the set of preimages $r^{-1}(\{z_{t+1}\})$ of the *phantasma* z_{t+1}. Together with such a transition $\langle z_t, z_{t+1} \rangle$, the agent experiences the cost c_{t+1} associated to the underlying transition $\langle x_t, x_{t+1} \rangle$ by the function C. Such cost is indeed a deterministic function determined univocally by $\langle x_t, x_{t+1} \rangle$. Nonetheless, for the agent that observes the system through a generic representation r, such a cost can at best be expressed as a stochastic quantity only partially determined by $\langle z_t, z_{t+1} \rangle \in U_r$. More precisely, on the sole basis of the knowledge that the current *phantasma* is z_t, the agent

might expect to observe a cost belonging to the following set:

$$\Big\{c \mid c = C(\langle x_i, x_j \rangle) \text{ with } x_i \in r^{-1}\big(\{z_t\}\big),\ x_j \in r^{-1}\big(\{z_{t+1}\}\big)\Big\}. \qquad (13)$$

In the following, we consider cases in which although a Markov representation of the underlying system can be easily obtained, it is convenient, for computational reasons, to consider a representation of non-Markov type. In particular, the concepts of Markov and non-Markov representations are the key elements for the definition of the class of algorithms that we call *ant programming* and that will be introduced in the next section. Among the possible different instances of this class of algorithms, the most significative are those that adopt a somehow extreme attitude towards Markovianity. On the one hand, *Markov ants* give, of the underlying problem at hand, a faithful representation that enables to find the optimal solution, but that for larger problems leads to intractability. On the other hand, *Marco's ants*, by adopting a non-Markov representation, are not guaranteed to find the optimal solution but handle a more compact representation graph and therefore in practice revealed to be much more suitable for larger problems.

4 Ant programming

In this section we introduce a new class of algorithms which deal with the optimization problems (3) under the form described by (10). The class of algorithms we introduce here is inspired by *ant colony optimization*, and from the latter it inherits the essential features, the terminology and the underlying philosophy.

The aim of this section is mostly speculative: we do not describe here a specific algorithm, but rather a class in the sense that we define a general resolution strategy and an algorithmic structure where some components are functionally specified but left uninstantiated.

In the following, with *ant programming* we will refer to this class of algorithms together with the collection of problems (3) or equivalently (10).

Ant programming is introduced here mainly as a mean to gain insight into the general principles underlying the use of a Monte Carlo approach for the solution of the class of problems (3). Such an insight will throw a new light on *ant colony optimization* itself.

4.1 The three phases of ant programming

Two are the essential features of *ant programming*. The first is the incremental Monte Carlo generation of complete paths over the state graph $\mathcal{G}$, on the basis of information contained in the representation graph $\mathcal{G}_r$. The second is the use of the cost of the generated solutions to bias subsequent generations of paths. These two features are described in terms of the three *phases* which, properly iterated,

constitute *ant programming*: At each iteration, a *cohort* of *ants* is considered. Each *ant* in the *cohort* undergoes a *forward* phase that determines the generation of a path, and a *backward* phase that states how the costs experienced along such a path should influence the generation of future paths. Finally, each iteration is concluded by a *merge* phase that combines the contribution of all the *ants* of the *cohort*.

The three phases *forward*, *backward*, and *merge* are in turn characterized by the three *operators* π_ϵ, ν, and σ respectively.

The following sections give a detailed analysis of these elements of *ant programming*.

4.1.1 Monte Carlo generation of paths: The "forward" phase

Using the terminology inherited from *ant colony optimization* and in the light of the formalization given in Section 3, *ant programming* metaphorically describes each Monte Carlo run as the walk of an *ant* over the graph $\mathcal{G}(X, U, C)$, where at each node a random experiment determines the following node. In the ant metaphor, the random experiment is depicted as a *decision* taken by the *ant* on the basis of a probabilistic policy parametrized in terms of information contained in the graph $\mathcal{G}_r(Z_r, U_r, T)$.

Let us give a detailed description of this decision process that will be hereafter denoted as the *forward* phase. Let us suppose that after t decision steps the partial solution built so far is $(u_0, \ldots, u_{t-1})$. The state of the solution generation process is therefore $x_t = (u_0, \ldots, u_{t-1})$. In the ant metaphor, this fact is visualized as an *ant* being in the node x_t of the graph $\mathcal{G}(X, U, C)$. The *ant* perceives the state x_t in terms of the *phantasma* $z_t = r(x_t)$ of the graph $\mathcal{G}_r(Z_r, U_r, T)$. In the general case, it is not possible to express the set $U_r(t)$ of admissible actions available to the *ant* when in z_t only in terms of z_t itself, and of the information contained in $\mathcal{G}_r$. Indeed, it is necessary the knowledge of the actual state x_t, and of the set $U(x_t)$ of the edges departing from node x_t in the graph $\mathcal{G}$. The set $U_r(t)$ of the admissible actions at time t is indeed:

$$U_r(t) = U_r(z_t|x_t) = \Big\{ \langle z_t, z_{t+1} \rangle \in U_r \;\Big|\; z_{t+1} \in r\big(U(x_t)\big), z_t = r(x_t) \Big\}.$$

The decision of the *ant* consists in the selection of one element from the set $U_r(z_t|x_t)$ of the available transitions, as described at the level of the graph $\mathcal{G}_r$. Once an element, say $\langle z_t, z_{t+1} \rangle$, is selected, the partial solution is transformed according to Eq. 4 and Eq. 9:

$$x_{t+1} = [x_t, u_t] = (u_0, \ldots, u_{t-1}, u_t),$$

where $x_{t+1} \in r^{-1}(\{z_{t+1}\})$ is one of the preimages of the *phantasma* z_{t+1}. If more than one preimage exist, one among them is arbitrarily selected. In terms of the metaphor, this state transition is described as a movement of the *ant* to the node

x_{t+1} of $\mathcal{G}$ which in turn is perceived by the *ant* as a movement to the *phantasma* $z_{t+1} = r(x_t)$ on $\mathcal{G}_r$.

The decision among the elements of $U_r(z_t|x_t)$ is taken according to the first operator of *ant programming*: the *stochastic policy* π_ϵ. Given the current *phantasma* and the set of admissible actions $U_r(z_t|x_t)$, the policy selects an element of $U_r(z_t|x_t)$ as the outcome of a random experiment whose parameters are defined by the weights $T(\langle z_t, z_{t+1}\rangle)$ of the edges $U_r(z_t|x_t)$ on the graph $\mathcal{G}_r(Z_r, U_r, T)$. Accordingly we will adopt the following notation to denote the stochastic policy:

$$\pi_\epsilon\big(z_t, U_r(z_t|x_t); T|_{U_r(z_t|x_t)}\big). \tag{14}$$

With the notation $T|_{U_r(z_t|x_t)}$ we want to suggest that, when in z_t, the the full knowledge of the function T is not strictly needed in order to select an element of the set $U_r(z_t|x_t)$. Indeed it is sufficient to know the restriction of T to the subset $U_r(z_t|x_t)$ of the domain U_r of T.[10]

In relation to the definition of the policy π_ϵ, it is worth noticing here how the decision process uses the information contained in the two graphs $\mathcal{G}$ and $\mathcal{G}_r$: The decision is taken on the basis of information pertaining to the graph $\mathcal{G}_r$, restricted by the knowledge of the actual state x_t which in turn is a piece of information pertaining to the graph $\mathcal{G}$.

For future reference, it is important to notice that T, that is, the weights of the graph $\mathcal{G}_r$, are to be intended as parameters of the policy π_ϵ: Changing the weights of $\mathcal{G}_r$ amounts therefore to changing the policy itself. Further, in the following the subscript ϵ will indicate, in a sense that will depend on the actual implementation of π_ϵ, the *degree of stochasticity* of the policy, such that $\epsilon = 0$ will mean a deterministic policy. For the moment, the reader can see the subscript as a mere reminder of the *possibly* stochastic nature of the policy π_ϵ.

Given the abstract definition (14) of the policy π_ϵ, the *forward* phase can be defined as the sequence of steps that take one *ant* from the initial state x_0, to a solution, say $s = x_\tau$, of the original combinatorial problem (3). Each of such steps is composed by the two operations of first selecting a transition on the graph $\mathcal{G}_r$, and then actually moving on the graph $\mathcal{G}$ from the current node x_t to the neighboring node x_{t+1}. Formally, the single *forward* step is described as:

$$\begin{aligned} \langle z_t, z'_{t+1}\rangle &= \pi_\epsilon\big(z_t, U_r(z_t|x_t); T|_{U_r(z_t|x_t)}\big); \\ x_{t+1} &= \mathcal{F}\big(x_t, \langle z_t, z'_{t+1}\rangle\big); \\ z_{t+1} &= r(x_{t+1}), \end{aligned} \tag{15}$$

where the operator π_ϵ is the stochastic policy that indicates the transition to be executed as seen on the graph $\mathcal{G}_r$, and where with the operator $\mathcal{F}$ we denote the

[10]This fact is the expression of one of the feature of *ant programming*, namely the *locality* of the information needed by the *ant* in order to take each elementary decision. Such a feature plays and important role in the implementation, allowing a *distribution* of the information on the graph of the representation $\mathcal{G}_r$.

operation of selecting one preimage x_{t+1} of z'_{t+1} and moving to it on the graph $\mathcal{G}$ from the current state x_t. Such a movement on $\mathcal{G}$ will be indeed perceived by the *ant* as a movement to the *phantasma* $z_{t+1} = r(x_{t+1}) = z'_{t+1}$, as requested by the policy π_ϵ.

4.1.2 Biasing the generation of paths: The "backward" and "merge" phases

The ultimate goal of *ant programming* is to find a policy $\bar{\pi}_\epsilon$, not necessarily stochastic, such that a sequence of decisions taken according to $\bar{\pi}_\epsilon$ leads an *ant* to define the solution $\bar{s}$ which minimizes the cost function J of the original optimization problem (3).

Since the generic policy (14) is described parametrically in terms of the function T, that is, in terms of the weights associated to the edges of the graph $\mathcal{G}_r$, a search in the space of the policies amounts to a search in the space of the possible weights of the graph $\mathcal{G}_r$ itself.

From a conceptual point of view, the function T is to be related to Hamilton's *principal function* of the calculus of variations, and to the *cost-to-go* and *value function* of dynamic programming and reinforcement learning. More precisely, the function T can be closely related to the function that in the reinforcement learning literature is known as "*state*-action value function," and that is customarily denoted by the letter Q.[11]

A complete analysis of how to express the Q function for the problem at hand in terms of the function T is, at this stage of our argumentation, not possible yet. For the time being, it will be sufficient to notice that $T(\langle z_t, z_{t+1} \rangle)$ determines, as to Eq. 14, the probability of selecting the action "go to *phantasma* z_{t+1}" when

[11] For the benefit of the reader who is not deeply acquainted with reinforcement learning theory, we note that a reference to the Q function does not necessarily imply a reference to Q-learning. The latter is a specific algorithm for learning a Q function. The Q function itself is the more general concept of a function that associates a number to a *state*-action pair, the number being the total future cost that would be incurred if, from the given *state*, one selects the given action and then behaves optimally thereafter.

A further terminological remark involves the reason why the word "state" is *italicized* in the previous sentence. As argued in Section 3.2, a state enjoys the Markov property by definition and therefore a "non-Markov state" is, in this sense, a contradiction in terms. Nevertheless, in the reinforcement learning literature it is customary to refer to the concept of "state" and to adopt the above mentioned definition of Q irrespectively, both when the Markov property holds, and when it does not.

In general, we regard this abuse of terminology as *bad practice*. Though usually this does not cause major harms, it introduces unnecessary inconsistencies between the literature in reinforcement learning on the one side, and "more classical" fields as control theory or system theory on the other. Because in this work we refer both to the level of the state description and to the level of the representation, it is particularly important to be strict with the distinction between the concept of *state* and the concept of *phantasma*. According to the terminology introduced in Section 3.3, the function Q is to be defined as $Q : Z \times U \to \mathbb{R}$. It therefore maps a *phantasma*-action pair into a number.

the current *phantasma* is z_t. It therefore associates to the *phantasma*-action pair, a number which represents the *desirability* of performing such an action in the given *phantasma*. In this respect, and taking into account the remark of Note 11, it is clear the similarity with the role of the function Q in reinforcement learning. Moreover, as it will be made clear presently, the value of $T(\langle z_t, z_{t+1}\rangle)$ is generally given as a *statistic* of the observed cost of paths containing the transition $\langle z_t, z_{t+1}\rangle$. It therefore brings information on the quality of the solution that can be obtained by "going to z_{t+1}" when in z_t. Also in this respect, it can be stated a parallel with the function Q which indeed informs on the long-term cost of a given action, provided that future actions are selected optimally.

In *ant programming*, as generally in reinforcement learning, the search in the space of the policies is performed through some form of *generalized policy iteration* [18]. Starting from some arbitrary initial policy, *ant programming* iteratively generates a number of paths in order to *evaluate* the current policy and then *improves* it on the basis of the result of the evaluation.

At each iteration, therefore, a *cohort* of *ants* is considered, each generating a solution through a *forward* phase as described in the previous section. Once the solution is completed, each *ant* traces back its path proposing a new value of the function T on the basis of the costs experienced in the forward movement—and possibly on the basis of the current value of T. This phase is denoted in the terminology of *ant programming* as the *backward* phase of the given *ant*. The actual new value of T is obtained by some combination of the values proposed by the *ants* of the *cohort*. This phase is denoted as the *merge* phase.

Let us now see in detail the *backward* phase for a given single *ant*. Let us consider a complete path $x = \langle x_0, x_1, \ldots, x_\tau\rangle$ over the graph $\mathcal{G}$. Let $s = x_\tau$ be the solution associated to the path x, and $c = \langle c_1, \ldots, c_\tau\rangle$ be the sequence of costs experienced along the path. Further, let $z = \langle z_0, z_1, \ldots, z_\tau\rangle$ be the path x as perceived by the *ant* under the representation r. That is, $z_t = r(x_t)$ with $t = 0, \ldots, \tau$.

The key element of the *backward* phase is the operator ν that uses the observed costs associated to the solution s in order to propose a new function T'. Similarly to the *forward* phase, the *backward* phase is composed by a sequence of steps, each formally described by the pair of operations:

$$\begin{aligned} z_t &= \mathcal{B}(z_{t+1}, z), \\ T'(\langle z_t, z_{t+1}\rangle) &= \nu(c, T), \end{aligned} \tag{16}$$

where with the operator $\mathcal{B}$ we indicate a single step backward on the graph $\mathcal{G}_r$, along the forward trajectory z. The operator ν proposes a new value for the weight associated to each visited edge $\langle z_t, z_{t+1}\rangle$, on the basis of the sequence of costs experimented during the *forward* phase, and of the current values of the function T.

It is intuitive that each single *ant* proposes values of T' for those transitions $\langle z_t, z_{t+1}\rangle$ that it has experienced along the path z, and leaves undetermined those

related to unseen transitions. Hence, in our pictorial description of *ant programming*, this phase is pictured through an *ant* that "traces back" its forward path and leaves on such a path some information.

From a logical point of view, the different strategies for propagating the information gathered along a path are to be related to the different *update* strategies in reinforcement learning. In particular, for an *ant* to propose values of T' only for the visited transitions and on the basis of the cost of the associated solution, is equivalent to what in reinforcement learning is called *Monte Carlo update* [18]. On the other hand, it is equivalent to a *Q-learning update* [19] to propose a value of T' for a visited transition on the basis of the experienced cost for the transition itself and of the minimum of the current values that T assumes on the edges departing from the node to which the considered transition leads.

The details of the definition of the *backward* phase, and in particular of the operator ν are not given as part of the description of *ant programming* and are left uninstantiated.

In the same spirit, we leave here undefined in its details also the *merge* phase in which it is performed a combination of the different functions T' proposed by the individual *ants* of the same *cohort*. At this level of our description it will be sufficient to note that, for every transition $\langle z_t, z_{t+1}\rangle \in U_r$, the actual new value of $T(\langle z_t, z_{t+1}\rangle)$ will be some linear or nonlinear function of the current value of $T(\langle z_t, z_{t+1}\rangle)$, and of the different $T'_j(\langle z_t, z_{t+1}\rangle)$, where j is the index ranging over the *ants* of the *cohort*. The *merge* phase will be therefore characterized by the operator σ:

$$T(\langle z_t, z_{t+1}\rangle) = \sigma\big(T(\langle z_t, z_{t+1}\rangle), T'_1(\langle z_t, z_{t+1}\rangle), T'_2(\langle z_t, z_{t+1}\rangle), \dots\big). \tag{17}$$

Different possible instances of the operators ν and σ will be discussed in a future work.

4.2 The algorithm and the metaphor

The abstract definition of *ant programming* has been given in the previous sections in terms of the operators π_ϵ, ν, and σ. In order to define an instance of the *ant programming* class, such operators need to be instantiated and defined in their details. Together with the operators π_ϵ, ν, and σ, the other key element in the definition of an instance of the class, is the generating function r that defines the relation between the state graph $\mathcal{G}$ and the representation $\mathcal{G}_r$. We will therefore denote an instance of *ant programming* with the 4-tuple

$$\langle r, \pi_\epsilon, \nu, \sigma\rangle. \tag{18}$$

Indeed, other elements are to be instantiated as, for instance, the way to select one preimage of a given *phantasma* in Eq. 15, or the number N of ants composing a *cohort* and the way of initializing the function T. Anyway, such elements are

either less relevant, or are to be defined as a more or less direct consequence of the definition of the 4-tuple (18). For the sake of clarity and readability of the notation, we will therefore adopt Expression 18 to denote an instance of *ant programming*. In future developments, if it will be necessary to make the distinction between two instances that differ only for elements other than those considered in the 4-tuple, Eq. 18 will be extended to include the elements that are needed in order to refer univocally to each of the instances under analysis.

In particular, the 4-tuple (18) gives an operative definition of the function T, which is the function that the algorithm iteratively modifies and refines with the aim of converging to an optimal policy. As seen in the previous sections, the generating function r, together with the graph $\mathcal{G}$, gives the topology of the graph $\mathcal{G}_r$ and determines therefore the domain of the function T. The operator π_ϵ defines how the values of T are used in the decision process, while the operators ν and σ define how the function T is to be modified on the basis of the quality of the solutions obtained.

Considered the key role of the function T in *ant programming*, we now provide this function with an interpretation in terms of the *ant metaphor*. At the same time, this will complete the pictorial description of the algorithm.

As anticipated in Section 3.3, the value $T(\langle z_t, z_{t+1}\rangle)$ associated to the edge $\langle z_t, z_{t+1}\rangle$ is called, in the ant metaphor, the value of the *pheromone trail* on the edge $\langle z_t, z_{t+1}\rangle$ itself. A single Monte Carlo run is described by an *ant* that walks on the graph $\mathcal{G}$, starting from the node x_0, and incrementally builds the solution s of cost $J(s)$. Each visited state x_t, is indeed perceived by the *ant* as the *phantasma* $z_t = r(x_t)$. At each time step, the following node x_{t+1} is selected by the procedure described by the *forward* step (15). Such procedure involves a *decision* taken by the *ant* according to the policy π_ϵ and therefore according to the value of the *pheromone trail* on the edges of $\mathcal{G}_r$, departing from the current *phantasma*.

Once the solution s is completed, the *ant* traces back the path that led to the solution itself, and *deposits its pheromone*. Out of the metaphor, the values of the function T are changed for those edges $\langle z_t, z_{t+1}\rangle$ that describe on the graph $\mathcal{G}_r$ the path leading to the solution s.

The role of the *pheromone trails* T is to make available the information gathered on a particular path by one *ant* belonging to one given *cohort*, to other *ants* of a future *cohort*; it is therefore a form of *inter-cohort* communication mediated by the graph $\mathcal{G}_r$. For real insects, the notion of *stigmergy* [16] has been introduced to denote a form of "stimulation [...] by the very performance [...] achieved." More in general, the term has been used to describe any indirect communication mediated by modification of the environment that can be observed in social insects [5, 8]. Accordingly, in the ant metaphor it is customary to refer to the interaction between *ants* with the term *stigmergy* [10].

As a way to give a summary of the concepts discussed in this section, we propose in Table 1 a *pseudo-code* description of *ant programming*.

Table 1: Pseudo-code description of *ant programming*.

```
/* initialize pheromone */
T = init-pheromone();
ad-libitum
do
   /* consider a cohort of N ants */
   for i = 1 to N
   do
      t = 0;   x_0 = ( );   z_0 = r(x_0);   z = ⟨z_0⟩;   c = ⟨ ⟩;

      /* forward phase */
      while x_t ∉ S
      do
         ⟨z_t, z'_{t+1}⟩ = π_ε(z_t, U(z_t|x_t); T|_{U(z_t|x_t)});
         x_{t+1} = F(x_t, ⟨z_t, z'_{t+1}⟩);

         c_{t+1} = C(⟨x_t, x_{t+1}⟩);   c = append(c, c_{t+1});
         z_{t+1} = r(x_{t+1});   z = append(z, z_{t+1});

         t = t + 1;
      od

      /* backward phase */
      while z_t ≠ z_0
      do
         z_{t-1} = B(z_t, z);
         T'_i(⟨z_{t-1}, z_t⟩) = ν(c, T);

         t = t - 1;
      od
   od

   /* merge phase */
   for-each ⟨z_i, z_j⟩ ∈ U_r
   do
      T(⟨z_i, z_j⟩) = σ(T(⟨z_i, z_j⟩), T'_1(⟨z_i, z_j⟩), T'_2(⟨z_i, z_j⟩), ..., T'_N(⟨z_i, z_j⟩));
   od
od
```

At this point, having defined the 4-tuple (18), we have completed the definition and the analysis of the essential elements that are necessary to handle the complexity of the combinatorial problem (3), in the spirit of the solution strategy originally suggested by *ant colony optimization*. In this sense, *ant programming*, beside defining a general strategy for solving the combinatorial problem (3), provides also a formal tool for gaining some further insight in the deep motivations of *ant colony optimization*.

In particular, *ant programming* defines the notion of a generic *generating function of the representation*, keeping therefore separated the problem at hand, from the problem representation that is assumed by the solution strategy. As a consequence, it becomes clear that the key element for understanding the properties of *ant colony optimization* is the the relation existing between state and *phantasma*, and between the graphs $\mathcal{G}$ and $\mathcal{G}_\tau$.

Even if the knowledge of the state is still necessary to the policy π_ϵ in order to check *on-line* the feasibility of the solution being built, the optimization of the decision process and the search for the optimal policy act on the function T and therefore refer only to the graph of the representation $\mathcal{G}_\tau$. If the generating function of the representation is not selected trivially as $r \equiv I$, the number of edges of the graph $\mathcal{G}_\tau$ might be dramatically smaller than the one of $\mathcal{G}$. In particular, for an appropriate choice of the function r, the number of the edges of $\mathcal{G}_\tau$ might grow polynomially with the number $|Y|$ of component of the original optimization problem, even if the number of edges of the graph $\mathcal{G}$ grows exponentially.

Actually, the above described process is the one carried out by *ant colony optimization*. There, the generating function r, thought not explicitly mentioned, is implicitly defined in a way that reduces dramatically the complexity of the graph on which the *ants* move. A preliminary analysis of the relation between *ant colony optimization* and the general *ant programming* method, is sketched in Section 5 with the definition of the specific instance of *ant programming* that we call *Marco's ants*.

5 Conclusions and future work

This paper addresses the definition of formal tools for the analysis of *ant colony optimization*, a class of algorithms for solving heuristically a discrete optimization problem with a finite number of solutions. To this aim the paper introduces the more general and abstract framework of *ant programming*.

Inspired by the foraging behavior of real ants, *ant colony optimization* proposes a heuristic solution to the problem at hand, through a process that involves iterated generations of paths on an appropriate graph. The generation of each path is described in terms of a metaphor by the walk of an *ant*, and represents the incremental construction of a feasible solution of the original combinatorial problem.

The incremental generation of solutions considered in *ant colony optimization* suggested a representation of the original combinatorial optimization problem in terms of a multi-stage decision process that can be throughly seen as a discrete-time optimal control problem. In turn, such an optimal control problem can be naturally mapped on the problem of finding paths of minimum cost on a graph whose nodes represent states of the controlled system. In other words, the incremental generation of a solution of the original combinatorial problem can be seen as the generation of a path on a graph.

The paper proposes a critical analysis of the concept of *state* of the process which incrementally constructs a solution of a combinatorial problem. Further, the paper draws a clear picture of the relation between concepts pertaining to this context and their natural counterparts in control theory. As a result of this analysis, it is of particular importance for future developments, the clear understanding that emerges from the paper on the notions of *representation* and of *Markov property* in relation to the class of combinatorial problems.

In particular, working on a Markov representation of an NP-hard problem is computationally infeasible. Therefore, the need emerges clearly for a computational method that generate a manageable representation, though this can be obtained only at the expense of the Markov property.

In order to give a well defined meaning to the previous sentence, the paper introduces the notion of *phantasma* as the piece of information, somehow related to the state, on the basis of which all the decisions are taken in the incremental construction of a solution.

In the paper, *ant programming* is introduced as an abstract class of algorithms inspired by *ant colony optimization*, and inheriting from it the essential features, the terminology, and the underlying philosophy.

Ant programming is an abstract class of algorithms in the sense that some of its elements are only functionally defined, and are left uninstantiated in their details. Indeed, *ant programming* has been introduced as a mean for gaining insight into the general principles underlying the use of a Monte Carlo approach for the incremental solution of a combinatorial optimization problem. Such an insight is intended to provide the designer of algorithms with new categories, an expressive terminology, and tools for dealing effectively with the peculiarities of the problem at hand.

Future work will concentrate on the analysis of *ant programming* and on the properties of its possible instances. In particular, it is of paramount importance to gain a full understanding of the impact of the choice of r, the *generating function of the representation*, on the resulting algorithms. Such a function associates a *phantasma* to the current state and therefore can be informally thought of as the "lens" under which the process of incremental construction of a solution is seen. In this sense, "the *ant* never thinks without a *phantasma*" and, as far as the decision process is concerned, this is to be understood as "the *ant* takes decisions on the basis of the *phantasma*."

The generating function determines therefore the information on the basis of which the decisions will be taken. At the extreme, the generating function might be a one-to-one mapping. In this case, only one state is associated to a *phantasma*, and *vice versa*. As a consequence, the state graph $\mathcal{G}$ and the representation graph $\mathcal{G}_r$ have the same topological structure and, as it can be easily shown, the representation enjoys the Markov property. Accordingly, we refer to this extreme instance of the *ant programming* class with the name of *Markov ants*.

Markov ants face directly the exponential explosion of the number of edges of the graph $\mathcal{G}$. Nevertheless, since r is a one-to-one mapping, no two states are *aliased* in the representation. As a consequence, the policy that according to (14) selects the action on the basis of the current *phantasma*, indeed implicitly bases the choice on the actual underlying state. From this fact, different appealing properties follow. It can be shown, for instance, that an optimal policy exists, and that it is deterministic.

The performance of *Markov ants* can be improved if the pheromone trails T and the operator ν are designed in such a way that the Markov property of the representation is fully exploited. This can be done by defining T as a costs-to-go function, and by allowing the operator ν to *bootstrap* [18]. In this way *Markov ants* would be basically reduced to an algorithm of the *temporal difference* class [18].

Anyway, *Markov ants* are not aimed to be implemented. The focus of *ant programming* is indeed on problems whose Markov representation is computationally intractable and, in such situations, *Markov ants* are ruled out by their own very nature. Still, *Markov ants* remain of great theoretical interest.

Another class of instances of *ant programming* is of much greater practical interest. In these instances, the function r is such that each *phantasma* is associated to one and only one of the possible solution components, and all the components are represented on the graph once and only once. Such a function r generates the representation $\mathcal{G}_r$ that is topologically equivalent to the graph that has been used in practically all the implementations of *ant colony optimization*, since its first "template" instance developed by Marco Dorigo and colleagues in 1991. Accordingly, we call *Marco's ants* the instances of this class.

Thanks to the concepts introduced in this paper, it becomes apparent that the representation graph $\mathcal{G}_r$ is here much more compact than the original graph $\mathcal{G}$. In spite of this, implementations of *ant colony optimization* have been shown to be comparable to or better than state-of-the-art tecniques on several difficult instances of NP-hard problems.

Moreover, under some "reasonable" assumptions on the characteristics of the other components of the algorithm, *ant colony optimization* has been proved to asymptotically converge in probability to the optimal solution [17].

In order to compensate the drastic lost of information due to the function r considered, most of the instances of *ant colony optimization* use some additional sources of information. Two major approaches have been followed. In the first

approach, some additional *a priori* knowledge about the problem at hand, has been combined to the estimate of the function T for the definition of the decision policy. In the second approach, local optimization procedures, *ad hoc* tailored on the problem at hand, have been used in order to improve the quality of the solutions generated by the *ants*.

Future developments of this work will analyze in detail the properties of the two above mentioned instances: *Markov ants* and *Marco's ants*.

Further, it will be of great practical interest to evaluate the possibility of designing other instances of *ant programming* that, on the one hand, keep an eye on the practical implementation, as *Marco's ants* do, and that, on the other one, try to preserve as much as possible the properties of a state-space representation, going therefore in the direction of *Markov ants*.

Always with the goal of designing effective algorithms, a future development of this work will address the issue of the optimal definition of the operators π_ϵ, ν, and σ, and of the other elements of the algorithm.

Moreover, as far as the theoretical analysis is concerned, *ant programming* will be discussed in the light of a comparison with established frameworks like dynamic programming, reinforcement learning, and heuristic search. From such an analysis we expect to gain a final and full understanding of the differences, the relative advantages, and the respective fields of applicability.

References

[1] R. Beckers, J. L. Deneubourg, and S. Goss. Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology*, 159:397–415, 1992.

[2] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, USA, 1995. Vols. I and II.

[4] V. Boltyanskii. *Optimal Control of Discrete Systems*. John Wiley & Sons, New York, NY, USA, 1978.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, USA, 1999.

[6] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behavior. *Nature*, 406:39–42, 2000.

[7] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[8] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[9] 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*. McGraw-Hill, 1999.

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

[11] M. Dorigo, G. Di Caro, and T. Stützle. Ant algorithms (editorial). Special Issue on Ant Algorithms, *Future Generation Computer Systems*, 16(8), 2000.

[12] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[13] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[14] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[15] B. Friedland. Observers. In W. S. Levine, editor, *The Control Handbook*, pages 607–618. CRC Press, Boca Raton, FL, USA, 1995.

[16] P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis et cubitermes sp.* La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[17] W. Gutjahr. A graph-based ant system and its convergence. Special issue on Ant Algorithms, *Future Generation Computer Systems*, 16(8), 2000.

[18] R. S. Sutton and A. G. Barto. *Reinforcement Learning. An Introduction.* MIT Press, Cambridge, MA, USA, 1998.

[19] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, United Kingdom, 1989.

[20] L. A. Zadeh and C. A. Desoer. *Linear System Theory.* McGraw-Hill Book Company, Inc., New York, NY, USA, 1963.