Appeared in *Ant Algorithms - Proceedings of ANTS 2002, Third International Workshop on Ant Algorithms, Brussels, Belgium, September 12–14, 2002, Springer-Verlag, Lecture Notes in Computer Science Vol. 2463, 2002*

Toward the formal foundation of Ant Programming

Mauro Birattari, Gianni Di Caro, and Marco Dorigo

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{mbiro,gdicaro,mdorigo}@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* and, more in general, on the principles underlying the use of an iterated Monte Carlo approach for the multi-stage solution of combinatorial optimization problems. *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* we adopt 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 such as state, representation, and sequential decision process under incomplete information.

1 Introduction

In the last decade, a number of algorithms inspired by the foraging behavior of ant colonies have been introduced for the approximate solution of combinatorial optimization problems (see [8,10,9] for extensive reviews). The framework of *ant colony optimization* [8,10] gave recently a first unifying description of (most of) these algorithms. Loosely speaking, *ant colony optimization* presents the following features. A graph is defined in a way that each solution of the combinatorial problem corresponds to at least one path on the graph itself. The weights associated to the edges are such that the cost of a path equals the cost of 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 incrementally generated in a Monte Carlo fashion, and the observed costs are used to bias the generation of further paths. This process is iterated with the aim of gathering information on the graph and of eventually producing 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 represented 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* in order to bias the generation of future paths toward better solutions.

The very possibility of obtaining better solutions by exploiting memory about solutions generated so far is the basic assumption of *ant colony optimization.* A further implicit assumption relates what this memory should consist in. In spite of the key role played in all implementations of *ant colony optimization*, this assumption was never critically discussed before: The formal definition of *ant colony optimization* [8] envisages, for each optimization problem, a unique way of defining the memory. To clarify this issue, let us consider an optimization problem whose solutions are expressed by *ant colony optimization* as a sequence of components. *Ant colony optimization* generates solutions in the form of paths in the space of such components. Memory is kept of all the observed transitions between components. A degree of desirability is associated to each transition depending on the quality of the solutions in which it occurred so far. While a new solution is being incrementally generated, a component y is included with a probability that is proportional to the desirability of the transition between the last component included and y itself. Even if the assumption seems natural that memory should be associated with pairs of solution components, as done in *ant colony optimization*, in this paper we maintain that this assumption is just a matter of choice. Indeed, this is only one of the possible *representations* of the solution generation process that can be adopted for framing information about solutions previously observed. As it will be clear in the following, this representation is neither optimal nor the most natural, provided that a correct analysis of the problem at hand is given. Our analysis will be based on a clear understanding of the concept of state of the process of incremental solution construction.

In this paper, we propose a novel formal description of the combinatorial optimization problems to which *ant colony optimization* applies, and we analyze the implication of adopting a generic solution strategy based on the incremental Monte Carlo construction of solutions biased by a memory. This paper is an abridged version of a previously unpublished work by the same authors [4] and complements other recent theoretical analysis [15,21,12]. The paper introduces *ant programming* as an abstract class of algorithms which presents the characterizing features of *ant colony optimization* but which is more amenable to theoretical analysis for what concerns the concepts of representation and state. In particular, *ant programming* bridges the terminological gap between *ant colony optimization* and the fields of optimal control [3] and reinforcement learning [17]. Accordingly, the name *ant programming* was chosen for its assonance with *dynamic programming*, with which *ant programming* has in common the stress on the concept of state and the related idea of reformulating an optimization problem as a multi-stage decision problem and then searching for a good (hopefully optimal) decision *policy* for the latter. Both in dynamic programming and in *ant programming*, such a reformulation is not trivial and requires an *ad hoc* analysis of the optimization problem under consideration. These concepts, being among the main issues in this research, will be discussed in detail in the rest of the paper: Section 2 shows how to reformulate a discrete optimization problem into a discrete-time optimal control problem and then into a shortest path problem. Section 3 introduces the concepts of *graph of the representation*, *phantasma* and *sequential decision process* under incomplete information. Section 4 introduces and discusses the *ant programming* abstract class of algorithms. Section 5 discusses the main issues and describes the future developments of our research.

2 Discrete optimization, optimal control, and shortest paths

Let us consider a discrete optimization problem defined by a finite set S of feasible solutions and by a cost function J. The set S is:

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

where each solution s_i is a n_i-tuple

$$s_i = (s_i^0, s_i^1, \dots, s_i^{n_i-1}), \quad n_i \in \mathbb{N}, \quad n_i \leq n < \infty, \tag{2}$$

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:

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

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.

A feasible solution in S can be built incrementally starting from the 0-tuple $x_0 = ()$, and adding one-at-a-time a component. The generic iteration can be described as:

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

where x_j is a *partial solution* of length j. A partial solution x_j is called *feasible* if it can be completed into a feasible solution $s_i \in S$, that is, if at least one feasible solution $s_i \in S$ exists, of which x_j is the initial sub-tuple of length j. It is understood that a process generating a sequence of feasible partial solutions necessarily ends up into a feasible solution. For each feasible partial solution x_j, we 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 (partial) solution x_{j+1}.

Now, the set X of all feasible tuples x_j is finite since both the set S and the length of each feasible solution s_i are finite. 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.

Since a feasible solution can be obtained incrementally, the original optimization problem can be reformulated as 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. Such a way of proceeding 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. Formally, a function $\mathcal{C} : X \setminus \{x_0\} \to \mathbb{R}$ must be conveniently defined, which associates a cost c_{j+1} to each tuple x_{j+1}.[1]

The finite-horizon multi-stage decision process described above can be thoroughly seen as a deterministic *discrete-time optimal control problem* [5]. The tuple x_j can be seen as the *state* at time $t = j$ of a discrete-time dynamic system whose state-transition application 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:

$$\begin{cases} x_{t+1} = [x_t, u_t], \\ y_{t+1} = u_t, \end{cases} \tag{5}$$

[1] Given rule (4), the tuple x_{j+1} determines uniquely the tuple x_j and the component u_j, and is in turn determined uniquely by them. Therefore, the function $\mathcal{C}$ 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 $\langle x_j, u_j, x_{j+1} \rangle$.

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

Now, let $\mathcal{U}$ be the set of all the admissible control sequences that bring the system from the initial state x_0 to a terminal state belonging to S: 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:

$$\bar{u} = \arg\min_{u \in \mathcal{U}} J\big(\mathcal{S}(u)\big), \tag{6}$$

where with "$\arg\min$" we denote the element of $\mathcal{U}$ for which the minimum of the composed function $J \circ S$ is attained. If such a minimum is attained for more than one element of $\mathcal{U}$, it is a matter of indifference which one is considered.

It is apparent that the solution of the problem of optimal control stated in (6) 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 uniquely 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 trajectories have a fixed maximum length n, all the possible state trajectories of the system (5) can be conveniently represented through a weighted and oriented graph with a finite number of nodes. Let $\mathcal{G}(X, U)$ be such a graph, where X is the set of nodes and U is the set of edges, and let $C : U \to \mathbb{R}$ be a function that associates a weight to each edge. In terms of system (5), each node of the graph $\mathcal{G}(X, U)$ 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. Finally, the function C is defined in terms of the function $\mathcal{C}$. Namely, $c_{t+1} = C(\langle x_t, x_{t+1} \rangle) = \mathcal{C}(x_{t+1})$ is the cost of the edge $\langle x_t, x_{t+1} \rangle$. Furthermore, on the graph $\mathcal{G}(X, U)$ we can single 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 $\mathcal{G}(X, U)$ and of the function C, the optimal control problem (6) 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.

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

3 Markov and non-Markov representations

Consistently with the optimal control literature, we have called *state* each node of the graph $\mathcal{G}(X, U)$ and, by extension, we call *state graph* the graph $\mathcal{G}$ itself. In the following, the properties of the state graph will be discussed in the perspective of the solution of problem (3), and in relation to the solution strategy of *ant colony optimization*. The ant metaphor will be used to visualize abstract concepts. In particular, we will picture the state evolution of system (5), and therefore the incremental construction of a solution, as the walk of an *ant* on the state graph $\mathcal{G}$. 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."

The state of a stochastic or deterministic dynamic system can be informally thought of as the piece of information that gives the most predictive description possible of the system at a given time instant.[2] Since what is known in the literature as *Markov property* is related precisely to the concept 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 is *intrinsically* Markov.[3] 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 structure in which an agent[4] frames experience: an agent refers to

[2] A detailed analysis of the concept of state in the context of ant colony optimization can be found in [4]. A general analysis of the concept of state is given in the classical literature on linear system theory [20], dynamic programming [2], and optimal control [3].

[3] For a discrete Markov decision process, the following holds by definition: $P(x_{t+1}|x^t, u^t) = P(x_{t+1}|x_t, u_t)$, where $x^t = (x_t, x_{t-1}, x_{t-2}, \dots)$ and $u^t = (u_t, u_{t-1}, u_{t-2}, \dots)$ indicate the past history of x and u, respectively. Now, let us consider a time-varying system whose state dynamic is given by $x_{t+1} = f_t(x_t, u_t, \xi_t)$ where x_t and u_t are respectively state and input at time t, and the state disturbance $\xi_t \sim P(\xi)$ is a white noise independent of the state and the input in the following sense: $P(\xi_t|x^t, u^t) = P(\xi_t)$. Clearly, x_{t+1} is a random variable whose distribution is $P(x_{t+1}|x_t, u_t) = P(\Xi_{x_{t+1}})$ where $\Xi_{x_{t+1}} = \{\xi : f_t(x_t, u_t, \xi) = x_{t+1}\}$ is the set of the values ξ that, for the given x_t and u_t, map to x_{t+1}, and $P(\Xi_{x_{t+1}})$ indicates the probability of observing a ξ belonging to such a set. The Markov property holds when the above introduced time-varying system is seen as a decision process. In particular:

$$P(x_{t+1}|x^t, u^t) = \sum_{\Xi_{x_{t+1}}} P(\xi|x^t, u^t) = \sum_{\Xi_{x_{t+1}}} P(\xi) = P(\Xi_{x_{t+1}}) = P(x_{t+1}|x_t, u_t).$$

The treatment given above assumes that ξ is a discrete variable. Thought the property holds also for continuous ξ, the proof for the general case involves a more complex notation and goes beyond the scope of this footnote.

Conversely, any discrete Markov decision process is a state description of a system in the Kalman sense. It is straightforward to verify that any $x_{t+1} \sim P(x_{t+1}|x_t, u_t)$ can be written in the form $x_{t+1} = f_t(x_t, u_t, \xi_t)$ where the dependence on time t accounts for the fact that in the definition of the Markov property the distributions at different temporal instants need not be the same. Since $x_{t+1} = f_t(x_t, u_t, \xi_t)$ is the classical form in which the state dynamic of a generic time-varying system can be given, the assertion is proved.

[4] By *agent* we mean any entity acting on or observing purposely the system at hand.

a representation for describing the state of the system, for possibly keeping memory of observed trajectories, and for performing predictions or control actions. In the limit, a representation might bear the same information as the state. In this case the Markov property holds for such representation. In the more general case, a representation is of non-Markov type, that is, it gives less information than the state. 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 a representation that brings less information than a state description. 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 are facing a *quality-complexity dilemma*.

In the context of *ant colony optimization*, as pointed out in Section 1, the basic assumption is that better solutions can be obtained by exploiting memory about previously generated ones. In this context, the discussion proposed above entails two major issues. First, for most combinatorial optimization problems of interest for which the state space grows exponentially with the size of the problem itself, it is clear that it is infeasible to gather and use memory about solutions in terms of a state description: it is very unlikely that a trajectory has exploitable superpositions with previously generated ones. Therefore, in *ant colony optimization* it is necessary to refer to a representation that reduces the information retained about the current state. This determines some sort of *aliasing* of distinct states which induces a criterion for *generalizing* previous experience. Second, as it will be made clear in the following, since a generic representation is non-Markov, it is not possible to generate feasible solutions on the basis of the sole representation. Therefore, it is necessary to refer to a state description in order to insure that a feasible solution be generated. These two issues, taken together, force to devise a strategy for the incremental generation of solution that on the one hand refers to a state description for guaranteeing feasibility, and on the other hand refers to a representation for optimizing the quality of the generated solution. The characteristics of the representation to be adopted reflect the design choice regarding the trade-off associated with the *quality-complexity dilemma*. *Ant programming* makes explicit the necessity to refer both to a representation and to a state description. Every step in the incremental construction of a solution consists of two sub-steps: first, a set of feasible candidate actions is defined on the basis of information pertaining to the state description; second, one of such candidates is selected on the basis of its desirability expressed in terms of the representation. In this sense, *Ant programming* introduces the categories needed for understanding some mechanisms already adopted in *ant colony optimization* such as, for instance, keeping and updating at each step the list of the components whose inclusion into the solution under construction would make the latter unfeasible. Such a list implicitly brings information about the state of the solution construction process.

For the class of problems discussed in this paper, a formal definition of a representation can be given with reference to the state graph $\mathcal{G}(X, U)$. We define the *representation graph* as the graph $\mathcal{G}_r(Z_r, U_r)$, where Z_r is the set of the nodes and U_r is the set of the edges. 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: every element $z_t \in Z_r$

has *at least* one preimage in X, but generally the preimage is not unique. The notation $r^{-1}(\{z_t\}) = \{x_\tau | r(x_\tau) = z_t\}$ indicates 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. In the following, we will call each $z_t \in Z_r$ a *phantasma,* adopting the term used by Aristotle with the meaning of *mental image*.[5] With 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,* that is, what is retained about the system at time t for optimization purposes.[6]

Thanks to the notion of *phantasma*, we can give a precise interpretation to the concept of representation in the context of the control problem (6). As we pointed out before, the state evolution of the system (5) can be described as the walk of an *ant* on $\mathcal{G}(X, U)$. Let us assume now that the *ant* visits in sequence the nodes $x_0, x_1, \dots, x_n$. The same sequence, under the representation induced by r, appears as a sequence $z_0, z_1, \dots, z_n$ where for each i, with $0 \leq i \leq n$, z_i is the *phantasma* of the state x_i, that is, $z_i = r(x_i)$. In the *ant* metaphor, we say that the *ant*, though moving on the state graph $\mathcal{G}(X, U)$, *represents* its movement on the representation graph $\mathcal{G}_r(Z_r, U_r)$. In control theory, the process that carries the state into what we call a *phantasma,* is related to the concept of *state-space reduction*.[7]

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 an edge $\langle x_i, x_j \rangle \in U$ exists on the state graph such that x_i and x_j are the preimages under r of z_i and z_j, respectively. Formally:

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

When the system is described through a generic representation r, the subset $U_r(t) \subset U_r$ of the admissible control actions at time t cannot usually be described in terms of the *phantasma* z_t alone, but needs for its definition the knowledge of the underlying state x_t. In other words, for the generic generating function r, the *phantasma* z_t does not bring the same information as the state x_t and therefore the corresponding representation is non-Markov. The adoption of a non-Markov representation is by no means free from complications. While on the graph $\mathcal{G}$ every (partial) path is a (partial) feasible solution and *vice versa*, on $\mathcal{G}_r$ this property does not hold anymore. As far as the

[5] Aristotle (384–322 BC) *De Anima*: “The soul never thinks without a mental image.”

[6] As an example, let us consider the case in which the set Z_r coincides with the set of solution components Y and $r : [x_t, u_t] \mapsto u_t$. This is the typical transformation adopted in the applications of *ant colony optimization* to the traveling salesman problem and to other combinatorial optimization problems. For this reason, such a transformation will be denoted in the following as r_{aco}.

[7] Yet, the result of a *state-space reduction* does not have a standard name in control theory and the various terms used always bring a direct reference to the concept of state: e.g. *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.

construction of feasible solutions is concerned, $\mathcal{G}$ is not therefore superseded by $\mathcal{G}_r$: As anticipated before, the graph $\mathcal{G}_r$ and the information stored on it are used for optimizing the construction of a solution while the graph $\mathcal{G}$ is used for guaranteeing feasibility. In any case, because of the loss of topological information induced by the transformation from $\mathcal{G}$ to $\mathcal{G}_r$ and since the optimization process is based on $\mathcal{G}_r$, in the general case only sub-optimal solutions will be obtained.

The parallel of the weight function C of $\mathcal{G}$ for the graph $\mathcal{G}_r$ cannot be defined in a straightforward manner for a generic r. Moreover, it results more useful to define the weights of the edges of the graph $\mathcal{G}_r(Z_r, U_r)$ 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. The definition of the function T will be given in Section 4.

4 Ant programming

In this section we introduce *ant programming* as a new class of algorithms that deal with the optimization problems (3) under the form described by (6). *Ant programming* 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 a specific algorithm, but rather a class of algorithms, in the sense that we define a general resolution strategy and an algorithmic structure where some components are functionally specified but left uninstantiated.

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 desirability information provided by the function T associated with the representation graph $\mathcal{G}_r$. The second is the update of the desirability information in $\mathcal{G}_r$ on the basis of the cost of the generated solutions and the use of such information to bias subsequent generations. These two features are described in terms of the three *phases* that, when properly iterated, constitute *ant programming*: At each iteration, a new set of *ants*, hereafter called a *cohort*, 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, ν, and σ respectively.

The forward phase. Using the terminology of *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 $\mathcal{G}(X, U)$, 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 parameterized in terms of the function T, usually called the *pheromone trail*, defined on the set of edges of the graph $\mathcal{G}_r(Z_r, U_r)$.

The *forward* phase can be described as follows: Let us suppose that after t decision steps the partial solution built so far is $(u_0, \dots, u_{t-1})$. The state of the solution generation process is therefore $x_t = (u_0, \dots, u_{t-1})$. In the ant metaphor, this fact is visualized as an *ant* being in the node x_t of $\mathcal{G}(X, U)$. The *ant* perceives the state x_t in terms of the *phantasma* $z_t = r(x_t)$. 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 given by $\mathcal{G}_r$. The set $U_r(t)$ of the admissible actions at time t is indeed:

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

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. 5: $x_{t+1} = [x_t, u_t] = (u_0, \dots, u_{t-1}, u_t)$, where $x_{t+1} \in r^{-1}(\{z_{t+1}\})$ is one of the preimages of the *phantasma* z_{t+1}. 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. 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 $T(\langle z_t, z_{t+1} \rangle)$ associated with the edges $U_r(z_t|x_t)$ of the graph $\mathcal{G}_r(Z_r, U_r)$. Accordingly we will adopt the following notation to denote the stochastic policy:

$$\pi\big(z_t, U_r(z_t|x_t); T|_{U_r(z_t|x_t)}\big). \tag{7}$$

With the notation $T|_{U_r(z_t|x_t)}$ we want to suggest that, when in z_t, the full knowledge of the function T is not strictly needed 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.[8] The function T plays the role of parameter of the policy π: changing T will change the policy itself.

In relation to the definition of the policy π, 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 $\mathcal{G}$.

Given the abstract definition (7) of the policy π, 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 three operations: first define, on the basis of the current state x_t, the set $U_r(z_t|x_t)$ of the available transitions; second select a transition on $\mathcal{G}_r$; and third move on $\mathcal{G}$ from the

[8] 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 $\mathcal{G}_r$.

current node x_t to the neighboring node x_{t+1}. Formally, the single *forward* step is described as:

$$\begin{aligned} \langle z_t, z'_{t+1}\rangle &= \pi\big(z_t, U_r(z_t|x_t); T|_{U_r(z_t|x_t)}\big); \\ x_{t+1} &= \mathcal{F}\big(x_t, \langle z_t, z'_{t+1}\rangle\big); \\ z_{t+1} &= r(x_{t+1}), \end{aligned} \tag{8}$$

where the operator π 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 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.

The backward phase. The ultimate goal of *ant programming* is to find a policy $\bar{\pi}$, not necessarily stochastic, such that a sequence of decisions taken according to $\bar{\pi}$ 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 (7) 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. In fact, $T(\langle z_t, z_{t+1}\rangle)$ determines, as to (7), the probability of selecting the action "go to *phantasma* z_{t+1}" when 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, it is clear the similarity with the role of the function Q in reinforcement learning.[9] 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* [17]. 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. Once the solution is completed, each *ant* traces back its path proposing at each visited *phantasma* an update of the local values of the function T on the basis of the costs experienced in the forward movement. This phase is denoted in the terminology of *ant programming* as the *backward* phase of the given *ant*. The actual

[9] An important difference is precisely that the function Q supposes a direct knowledge of the state, while T refers to the *phantasma*. In reinforcement learning, the situation in which more states are not perceived as distinct is termed *perceptual aliasing* [19].

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 $x = \langle x_0, x_1, \dots, x_\tau \rangle$ over the graph $\mathcal{G}$. If $z = \langle z_0, z_1, \dots, z_\tau \rangle$ is the complete forward path as seen under r, and $c = \langle c_1, \dots, c_\tau \rangle$ is the experienced sequence of costs, then the single step of the *backward* phase is:

$$\begin{aligned} z_t &= \mathcal{B}(z_{t+1}, z), \\ T'(\langle z_t, z_{t+1} \rangle) &= \nu(c, T), \end{aligned} \tag{9}$$

where the operator $\mathcal{B}$ indicates a single step backward on $\mathcal{G}_r$, along the forward trajectory z. The operator ν is the key element of the *backward* phase. It has the role of proposing 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. 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, 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* [17]. On the other hand, it is equivalent to a *Q-learning update* [18] 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.

The merge phase. In the same spirit, we leave here undefined in its details also the *merge* phase which combines 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:

$$T(\langle z_t, z_{t+1} \rangle) = \sigma\big(T(\langle z_t, z_{t+1} \rangle), T'_1(\langle z_t, z_{t+1} \rangle), T'_2(\langle z_t, z_{t+1} \rangle), \dots\big). \tag{10}$$

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* was given in previous sections in terms of the operators π, ν, and σ. In order to define an instance of the *ant programming* class, such operators need to be instantiated. Together with the operators π, 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 $\mathcal{I} = \langle r, \pi, \nu, \sigma \rangle$. Indeed, other elements are to be instantiated as, for example, the number 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 $\mathcal{I}$.

In particular, the 4-tuple $\mathcal{I}$ gives an operative definition of the function T. 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 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. According to the pictorial description of *ant programming*, the function T is called *pheromone trail* and defines the policy π followed by the *ant* during the forward walk. Once a solution s is completed, the *ant* traces back its forward path and *deposits its pheromone* to update the function T. The role of the *pheromone trails* T is therefore 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 $\mathcal{G}_r$. ¿From the terminology adopted in the studies on social insects [13], it is customary to refer to such indirect communication with the term *stigmergy* [7].

At this point, having defined the 4-tuple $\mathcal{I}$, we have completed the definition of the 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*.

5 Discussion and future work

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 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, therefore, 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 (7) 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* [17]. In this way *Markov ants* would reduce to an algorithm of the *temporal difference* class [17]. Anyway, *Markov ants* are not meant 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 very own nature. Still, *Markov ants* remain of great theoretical interest.

Another class of instances of *ant programming* is of much greater practical interest. These instances are characterized by the function r_{aco}, as in Footnote 6, that associates a *phantasma* with one and only one of the possible solution components. The function r_{aco} generates the representation used in almost all the implementations of *ant colony optimization* since the first "template" instance developed by Marco Dorigo and colleagues [11,6] back 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 generated by r_{aco} is much more compact than the state graph. In order to compensate this drastic loss of information, most of the instances of *ant colony optimization* adopt some additional device both to guarantee the feasibility and to improve the quality of the solutions being built. As far as feasibility is concerned, all instances of *ant colony optimization* use an implicit description of the state graph usually in the form of a list of components already included into the solution under construction. As far as quality is concerned, 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*. Some of the resulting implementations have been shown to be comparable to or better than state-of-the-art techniques on several NP-hard problems. Moreover, under "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 [14,16].

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, try to preserve as much as possible the properties of a state-space representation, going therefore in the direction of *Markov ants*.

Acknowledgments: Mauro Birattari acknowledges support from the Metaheuristics Network, a Research and Training Network funded by the Commission of the European Communities under the Improving Human Potential programme, contract number HPRN-CT-1999-00106. The work of Gianni Di Caro has been supported by a Marie Curie Fellowship of the European Community programme Improving the Human Research Potential under contract number HPMF-CT-2000-00987. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate.

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. M. Birattari, G. Di Caro, and M. Dorigo. For a formal foundation of the Ant Programming approach to combinatorial optimization. Part 1: The problem, the representation, and the general solution strategy. Technical Report TR-H-301, ATR Human Information Processing Research Laboratories, Kyoto, Japan, 2000.
5. V. Boltyanskii. *Optimal Control of Discrete Systems*. John Wiley & Sons, New York, NY, USA, 1978.
6. M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.
7. M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.
8. 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, New York, NY, USA, 1999.
9. M. Dorigo, G. Di Caro, and T. Stützle (Editors). Special issue on “Ant Algorithms”. *Future Generation Computer Systems*, 16(8), 2000.
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, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.
12. M. Dorigo, M. Zlochin, N. Meuleau, and M. Birattari. Updating ACO pheromones using stochastic gradient ascent and cross-entropy methods. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and R. Raidl, editors, *EvoCOP 2002: Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 21–30. Springer-Verlag, Heidelberg, Germany, 2002.
13. 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.
14. W. Gutjahr. A graph-based ant system and its convergence. Special issue on Ant Algorithms, *Future Generation Computer Systems*, 16(8):873–888, 2000.
15. N. Meuleau and M. Dorigo. Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2):103–121, 2002.
16. T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4), 2002, in press.
17. R. S. Sutton and A. G. Barto. *Reinforcement Learning. An Introduction*. MIT Press, Cambridge, MA, USA, 1998.
18. C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King’s College, Cambridge, United Kingdom, 1989.
19. S. D. Whitehead and D. H. Ballard. Learning to perceive and act. *Machine Learning*, 7(1):45–83, 1991.
20. L. Zadeh and C. Desoer. *Linear System Theory*. McGraw-Hill, New York, NY, USA, 1963.
21. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-base search for combinatorial optimization. Technical Report TR/IRIDIA/2001-15, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2001.