Ant Colony Optimization and Stochastic Gradient Descent

Nicolas Meuleau
Marco Dorigo
IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, CP 194/6,
B-1050 Brussels, Belgium
nmeuleau@iridia.ulb.ac.be
mdorigo@ulb.ac.be

Abstract In this article, we study the relationship between the two techniques known as ant colony optimization (ACO) and stochastic gradient descent. More precisely, we show that some empirical ACO algorithms approximate stochastic gradient descent in the space of pheromones, and we propose an implementation of stochastic gradient descent that belongs to the family of ACO algorithms. We then use this insight to explore the mutual contributions of the two techniques.

Keywords
heuristic, ant system, ant colony optimization, combinatorial optimization, stochastic gradient descent, reinforcement learning

1 Introduction

The study of self-organization in social insects as a source of inspiration for novel distributed forms of computation is a promising area of AI known as *ant algorithms* (or sometimes as *swarm intelligence*) that is experiencing growing popularity [3, 4, 8, 11, 13]. A particularly successful form of *ant algorithm* is that inspired by ant colony foraging behavior. In these algorithms, applied to combinatorial optimization problems, a number of artificial ants are given a set of simple rules that take inspiration from the behavior of real ants. Artificial ants are then left free to move on an appropriate graph representation of the considered problem: they probabilistically build a solution to the problem and then deposit on the graph some artificial pheromones that will bias the probabilistic solution construction activity of future ants. The amount of pheromone deposited and the way it is used to build solutions are such that the overall search process is biased toward the generation of approximate solutions of improving quality.

The historic first example of an algorithm inspired by ant foraging behavior is the ant system (AS) [7, 14] and its first application was to the traveling salesman problem (TSP), a well known NP-hard problem [21]. As a follow-up of AS, a number of similar algorithms, each one trying either to improve performance or to make AS better fit a particular class of problems, were developed. Currently, many successful applications of such algorithms exist for $\mathcal{NP}$-hard academic combinatorial optimization problems such as quadratic assignment [18, 23], sequential ordering [16], resource-constrained project scheduling [24], vehicle routing with time windows [17], routing in packet-switched networks [6], shortest common supersequence [25], and frequency assignment [22]. Applications to real-world combinatorial optimization problems are starting to appear: For example, a gasoline distribution company in Switzerland is using ACO algorithms to choose routes of its trucks [17], while Fujitsu-Siemens Computers in Germany is testing ant colony optimization (ACO) for an important logistic problem [R. Palm, personal communication]. As a consequence, the ACO metaheuristic was recently defined [9, 10] to put in a common framework all the algorithms that can be considered as offspring of AS.[1]

[1] A notable exception is the ABC algorithm for routing that, although belonging to ACO, was developed independently of AS [31].

Despite these successes, the basic mechanisms at work in ACO are still loosely understood, and there is usually no analytical tool to explain the observed effectiveness of an ACO algorithm. In this article, we make a step in the direction of providing such tools by formally relating the ACO metaheuristic and the technique known as stochastic gradient descent (SGD), which has been extensively used in machine learning [26, 29]. More precisely, we show that some ACO algorithms approximate gradient descent of the expected value of the solution produced by an ant, in the space of pheromone trails. Moreover, we present an algorithm for combinatorial optimization that is, at the same time, an SGD algorithm working in the space of pheromones, and an instance of the ACO metaheuristic. This algorithm is an instance of the gradient-based reinforcement learning algorithm known as REINFORCE [35]. It can be seen both as a distributed, stigmergic[2] implementation of SGD, or as an ACO algorithm where the overall effect of ants' activity is to descend the gradient of a given function in the space of pheromones.

The interest of establishing connections between ACO and SGD is that it offers many opportunities of cross-fertilization. On one side, many questions asked in the study of ACO algorithms receive a second look under the gradient-descent interpretation. For instance, a new way of understanding and proving convergence of ACO algorithms is proposed. Moreover, some classical acceleration techniques for gradient-based reinforcement learning can be easily transposed to ACO algorithms. On the other side, ACO algorithms show how to implement effectively the technique of SGD for solving large combinatorial optimization problems. Several improvements to the basic trial-and-error search of artificial ants developed in the best ACO algorithms suggest, in turn, new ways of using gradient descent in the framework of combinatorial optimization.

In this introductory article, we present our main arguments using the example of the asymmetric TSP (ATSP). First (Sect. 2.1), we briefly review the AS algorithm using the ATSP as the example problem, and we show that AS is indeed closely related to the technique of SGD (Sects. 2.2 and 2.3). We then describe an implementation of SGD—or, more precisely, of William's REINFORCE [35]—that belongs to the family of ACO algorithms (Sect. 2.4). In Section 2.5, we show how to generalize this reasoning to any combinatorial optimization problem that can be solved by an ACO algorithm. Finally, we comment on these results and outline some future research directions in Section 3.

2 Ant System and Stochastic Gradient Descent

Ant system is a simple distributed algorithm that can be applied to any (constrained) minimum cost path problem on a graph. Throughout this article, we use the application of AS to the ATSP as a basic example to present our arguments.

2.1 Ant System

The ATSP can be defined as follows. Let X be a set of cities, $|X| = n$, and $D = [d(x, y)]$ be a distance matrix, with $d(x, y) \in \mathbb{R}^+$ for all $(x, y) \in X^2$. We will denote by $\bar{X}^t \subseteq X^t$ the set of acyclic paths of length $t \in \{1, \dots, n\}$ in terms of the number of cities crossed ($\bar{X}^1 = X$ and $\bar{X}^2 = \{(x, y) \in X^2\colon\ x \neq y\}$). ATSP can be defined as the problem of finding a path $h_n = (x_1, x_2, \dots, x_n) \in \bar{X}^n$ that minimizes the length of the corresponding tour, defined as

$$L(h_n) = \sum_{t=1}^{n-1} d(x_t, x_{t+1}) + d(x_n, x_1).$$

2 Stigmergy is a particular form of indirect communication used by social insects to coordinate their activities. Its role in the definition of ant algorithms is discussed at length in [3, 4, 8, 10].

The main variables of the AS algorithm are the pheromone trails $\tau(x, y)$ associated with each pair of cities $(x, y) \in \bar{X}^2$. Let $\mathcal{T}$ be the bidimensional vector gathering all the $\tau(x, y)$'s. The basic principle of AS is to simulate artificial *ants* that use the pheromone trails to build a random tour. Once its tour is completed, each ant makes a backward trip following the same path and updates the pheromones on its way back. Finally, the pheromone trails partially evaporate, that is, they decrease by a constant factor ρ, $0 < \rho \leq 1$, called the *evaporation rate*. The behavior of each ant can be summarized as follows:

Forward:

- Draw the start city x_1 at random uniformly;
- At each step $t \in \{1, \dots, n-1\}$, after following the path $h_t = (x_1, x_2, \dots, x_t) \in \bar{X}^t$, draw the next city at random following

$$\Pr(x_{t+1} = x \mid \mathcal{T}, h_t) = \begin{cases} 0 & \text{if } x \in h_t, \\ \tau(x_t, x) / \sum_{\substack{y \in X \\ y \notin h_t}} \tau(x_t, y) & \text{otherwise,} \end{cases} \tag{1}$$

  where $x \in h_t$ means that the acyclic path h_t traverses x.

Backward:

- After generating the path $h_n = (x_1, x_2, \dots, x_n) \in \bar{X}^n$, reinforce the pheromone trails $\tau(x_t, x_{t+1})$ for each $t \in \{1, \dots, n-1\}$ and $\tau(x_n, x_1)$ by the amount $1/L(h_n)$.

There are many ways of implementing the algorithm. In the original implementation of AS, a set of m artificial ants synchronously built m solutions as follows: First, all the ants perform their forward trip without updating the pheromones, and then all of them execute their backward trip and update the pheromones for the next "generation" of m ants. A pheromone evaporation stage takes place at each generation, before sending the ants backward. The total update at each generation of each pheromone $\tau(x, y)$, $(x, y) \in \bar{X}^2$ is then

$$\Delta\tau(x, y) = \left(\sum_{i=1}^{m} \frac{\delta_{x,y}(h_n^i)}{L(h_n^i)} \right) - \rho\tau(x, y),$$

where $h_n^i \in \bar{X}^n$ is the path followed by the ith ant during its forward trip ($i \in \{1, 2, \dots, m\}$), and $\delta_{x,y}(h_n) = 1$ if y is the immediate successor of x in the tour associated to $h_n \in \bar{X}^n$ and 0 otherwise. When $m = 1$, the different ants are sent one after the other in a fully sequential way, waiting for the previous ant to complete its backward trip before sending a new one. In this case, the pheromone update implemented by each ant is, for all $(x, y) \in \bar{X}^2$,

$$\Delta\tau(x, y) = \frac{\delta_{x,y}(h_n)}{L(h_n)} - \rho\tau(x, y), \tag{2}$$

where $h_n = \{x_1, x_2, \dots, x_n\} \in \bar{X}^n$ is the path followed by the ant during its forward trip. This pheromone update rule may also be used in a fully asynchronous and parallel implementation of ant system in which ants act completely independently of each other, and a pheromone evaporation stage is associated with each ant. Equation 2, originally introduced in [7, 14], is often replaced by the following rule, introduced for the first time in [12], in which the reinforcement of pheromone trails is multiplied by the evaporation rate ρ:

$$\Delta\tau(x, y) = \rho\frac{\delta_{x,y}(h_n)}{L(h_n)} - \rho\tau(x, y) = \rho\left(\frac{\delta_{x,y}(h_n)}{L(h_n)} - \tau(x, y)\right). \qquad (3)$$

Ant system was extensively tested together with a few other algorithms inspired by real ants' behavior on the TSP [14]. Although AS did not compete with the best known algorithms for TSP, its relative success inspired a great number of algorithms for different combinatorial optimization problems (cf. Introduction). Often the AS-based algorithms provide state-of-the-art performance. These algorithms have recently been put in a unifying framework called an ACO metaheuristic [9, 10]. ACO is composed of three main procedures. In the first one, artificial ants probabilistically construct feasible solutions to the considered problem by moving on a proper graph representation. In this phase the construction process is biased by previous experience memorized in the form of pheromone trails, and, in some implementations, by heuristic information available about the considered problem (see discussion in Sect. 3.1.1). The second phase, briefly discussed in Section 3.1.2, is optional: Here the solutions generated by the artificial ants can be taken to their local optima by a suitable local search routine. In the last phase, pheromone trails are updated by the ants, pheromone evaporation, and/or other suitable processes.

2.2 Stochastic Gradient Descent

ACO algorithms are usually regarded as optimization techniques working in the solution set of the combinatorial problem at hand. For instance, the ant system for ATSP is usually seen as an algorithm that tries to find a tour of minimal length (i.e., an optimal solution of the combinatorial problem). However, we adopt in this work a different approach and we look at ACO algorithms as working in the space of pheromone trails. In other words, we aim at finding *an optimal set of pheromones*, which can be defined in different ways. In this article, we focus on a particular form of optimality for pheromone values. We will call an optimal set of pheromones a configuration that optimizes *the expected value of the solution produced by an ant during its forward trip.* We then study how this problem may be solved using gradient descent in the continuous space of pheromone trails.

In the case of ATSP, we aim at maximizing the expected value of the inverse length of an ant's forward trip, given the current pheromone trails and the city-selection rule of Equation 1. That is, we will *climb* stochastically the gradient of the "error" $\mathcal{E}$ defined as[3]

$$\mathcal{E} \overset{\text{def}}{=} \mathrm{E}\left[\frac{1}{L(h_n)} \mid \mathcal{T}\right] = \sum_{h_n \in \bar{X}^n} \Pr(h_n \mid \mathcal{T})\frac{1}{L(h_n)}.$$

Note that the expectation is conditional on $\mathcal{T}$ because the probability of a given tour happening depends on the current pheromone trail vector $\mathcal{T}$, while the "local error"

3 Although we are dealing with maximization problems using stochastic gradient *ascent*, we use in this article the vocabulary associated with gradient *descent* algorithms, which is much more common in the machine learning literature (see the example of artificial neural networks in Sect. 3.2.1). The variable $\mathcal{E}$ represents the objective function of the gradient ascent algorithm (and not of the original combinatorial optimization problem), and it must be maximized. However, it plays the same role as the error function used in gradient descent algorithms.

$1/L(h_n)$ does not depend on the weights $\tau(x, y)$. Then we have

$$\frac{\partial \mathcal{E}}{\partial \tau(x, y)} = \sum_{h_n \in \bar{X}^n} \frac{\partial \Pr(h_n \mid \mathcal{T})}{\partial \tau(x, y)} \frac{1}{L(h_n)},$$

for each pair of cities $(x, y) \in \bar{X}^2$.

The probability of a given path is equal to the product of the probabilities of all the elementary events that compose it: if $h_n = (x_1, x_2, \ldots, x_n) \in \bar{X}^n$, then

$$\Pr(h_n \mid \mathcal{T}) = \prod_{t=1}^{n} \Pr(x_t \mid \mathcal{T}, h_{t-1}),$$

where h_t is equal to h_n truncated after step t: $h_t = (x_1, x_2, \ldots, x_t) \in \bar{X}^t$, and h_0 is the empty sequence. Therefore,

$$\frac{\partial \Pr(h_n \mid \mathcal{T})}{\partial \tau(x, y)} = \Pr(h_n \mid \mathcal{T}) \sum_{t=1}^{n} \frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)}.$$

Here we have supposed that $\Pr(x_t \mid \mathcal{T}, h_{t-1}) > 0$, which is always true because $x_t \notin h_{t-1}$, as h_t is an acyclic path; and because the pheromone trails never fall to 0 in the original AS algorithm (however, we will see later that this is a problem for the new algorithm). Define the *eligibility trace*[4] of (x, y) in path h_n as

$$T_{x,y}(h_n) \overset{\text{def}}{=} \frac{\partial \ln(\Pr(h_n \mid \mathcal{T}))}{\partial \tau(x, y)} = \sum_{t=1}^{n} \frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)}, \tag{4}$$

then

$$\frac{\partial \mathcal{E}}{\partial \tau(x, y)} = \sum_{h_n \in \bar{X}^n} \Pr(h_n \mid \mathcal{T}) \frac{T_{x,y}(h_n)}{L(h_n)} = \mathrm{E}\left[\frac{T_{x,y}(h_n)}{L(h_n)} \mid \mathcal{T}\right]. \tag{5}$$

We will see later how to calculate the traces $T_{x,y}$. We can already outline the basis of the SGD algorithm. Climbing the gradient of $\mathcal{E}$ corresponds to updating $\mathcal{T}$ iteratively in the direction of the gradient of $\mathcal{E}$:

$$\Delta\mathcal{T} = \alpha \nabla_{\mathcal{T}} \mathcal{E},$$

that is,

$$\Delta\tau(x, y) = \alpha \frac{\partial \mathcal{E}}{\partial \tau(x, y)},$$

for each individual "weight" $\tau(x, y)$, where $\alpha > 0$ is the step-size parameter or *learning rate.* Following Equation 5, we could do *exact* gradient ascent in the space of pheromone trails by enumerating all possible paths h_n and calculating, for each of them, the probability $\Pr(h_n \mid \mathcal{T})$ that an ant follows this path during its forward trip

4 We borrow this vocabulary from reinforcement learning literature. It is used in a similar gradient-based algorithm for optimal control of Markov decision processes [1, 35]. The *eligibility* of a weight is a measure of how much this weight will be involved in the next update. The $T_{x,y}$ variables are called *traces* because they keep track of the eligibility of the weights at each step t.

(given the current pheromone trails), the length of the corresponding tour $L(h_n)$, and the variable $T_{x,y}(h_n)$ for each $(x, y) \in \bar{X}^2$. Obviously, this approach would make no sense in practice: Once we have enumerated all possible paths we can solve our original problem simply by taking the best. Moreover, the size of $\bar{X}^n$ grows exponentially with the number of cities, so that this approach quickly becomes infeasible. Finally, the exact gradient descent performs poorly in many complex domains because it gets trapped in the first local optimum on its way.

In *stochastic* gradient descent (SGD), an unbiased random estimate of the gradient is used instead of the true gradient. In the case of our application to ATSP, Equation 5 shows that the gradient of $\mathcal{E}$ is the expected value of the random variable $T_{x,y}/L$ given the current pheromones (and the selection rule of Equation 1). Therefore, if we draw independently m paths $h_n^1, h_n^2, \ldots, h_n^m$ in $\bar{X}^n$ following the probability $\Pr(h_n \mid \mathcal{T})$, and average their *contributions* $T_{x,y}(h_n^i)/L(h_n^i)$ to the gradient, then the result

$$\frac{1}{m}\sum_{i=1}^{m}\frac{T_{x,y}(h_n^i)}{L(h_n^i)}$$

is a random vector whose expected value is equal to the gradient. In other words, it is an unbiased estimate of the gradient. This is true regardless of the number of paths sampled, even if only one sample is used to estimate the gradient (i.e., $m = 1$). The resulting stochastic algorithm has a reasonable complexity.[5] Moreover, it may escape from some low-value local optima on its path. It sometimes makes bad moves because the gradient estimate is wrong, but these moves may allow jumping out of a bad local optimum. Therefore SGD usually performs better than the exact gradient descent in large, highly multimodal search spaces.

The basis of our comparison between ACO and SGD is the analogy between the actions of sending an ant forward and sampling a tour h_n from $\Pr(h_n \mid \mathcal{T})$. During its forward trip, the action of an ant is precisely to sample a solution following this probability distribution. Therefore, the forward component of AS can be used in an SGD algorithm as well, and we just have to change the weight update rules. We show below that the updates associated with a given sampled tour are very similar in the two algorithms.

2.3 A first ACO/SGD Algorithm

In our ACO/SGD algorithm, each artificial ant samples a tour $h_n \in \bar{X}^n$ using the current pheromones and then updates every pheromone $\tau(x, y)$ following

$$\Delta\tau(x, y) = \alpha\frac{T_{x,y}(h_n)}{L(h_n)} \tag{6}$$

In a synchronous implementation, artificial ants are sent by groups of m ants that sample m tours without updating the pheromones (i.e., following the same probability distribution $\Pr(h_n \mid \mathcal{T})$). Then, each of them updates the pheromones adding a quantity of pheromone function of the quality of the solution it generated during its forward trip. The total amount of pheromone added by the m ants is then

$$\Delta\tau(x, y) = \alpha\sum_{i=1}^{m}\frac{T_{x,y}(h_n^i)}{L(h_n^i)},$$

5 We may actually influence the complexity by fixing the number m of samples drawn to calculate the estimate of the gradient. More samples allow a more accurate estimate.

for all (x, y), where $h_n^i \in \bar{X}^n$ is the path followed by the ith ant. This corresponds to using m independent samples h_n to calculate the gradient estimate (note that the factor $1/m$ has been absorbed in the learning rate α). The particular case $m = 1$ that is based on Equation 6 corresponds to a fully sequential implementation where only one sample is used to estimate the gradient and make a step in the space of pheromones. Equation 6 may also be used as the basis of a fully asynchronous and parallel implementation of the algorithm where the artificial ants act completely independently of each other. In this case, the gradient estimates may be slightly biased by the simultaneous read and write activity of the different ants. However, this bias will probably be negligible in many applications, provided that the learning rate α stays within reasonable bounds.[6]

Given a path $h_n = (x_1, x_2, \ldots, x_n) \in \bar{X}^n$ and a pair of cities $(x, y) \in \bar{X}^2$, the problem is now to calculate $T_{x,y}(h_n)$ as defined by Equation 4. Using Equation 1, we see that

- if $x \neq x_{t-1}$ or $y \in h_{t-1}$, then $\Pr(x_t \mid \mathcal{T}, h_{t-1})$ is independent of $\tau(x, y)$ and

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} = 0;$$

- if $x = x_{t-1}$ and $y = x_t$ then

$$\begin{aligned}
\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} &= \frac{\partial \ln\left(\tau(x, y)/\sum_{y' \notin h_{t-1}} \tau(x, y')\right)}{\partial \tau(x, y)}, \\
&= \frac{\partial \ln(\tau(x, y))}{\partial \tau(x, y)} - \frac{\partial \ln\left(\sum_{y' \notin h_{t-1}} \tau(x, y')\right)}{\partial \tau(x, y)}, \\
&= \frac{1}{\tau(x, y)} - \frac{1}{\sum_{y' \notin h_{t-1}} \tau(x, y')}, \\
&= \frac{1 - \Pr(y \mid \mathcal{T}, h_{t-1})}{\tau(x, y)};
\end{aligned}$$

- if $x = x_{t-1}$, $y \neq x_t$, and $y \notin h_{t-1}$ then, similarly,

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} = -\frac{\Pr(y \mid \mathcal{T}, h_{t-1})}{\tau(x, y)}.$$

Therefore, the SGD algorithm can be implemented using distributed, local information, as done in AS. The weight update corresponding to a sampled tour can be performed by the ant that sampled this tour during a backward trip. When returning to the tth city of the tour,[7] the ant updates the pheromone trail $\tau(x_t, x)$ for all $x \in X$ following

$$\Delta\tau(x_t, x) = \alpha \frac{1}{L(h_n)} \frac{1}{\tau(x_t, x)} (\delta(x_{t+1} = x) - \Pr(x \mid \mathcal{T}, h_t)), \tag{7}$$

6 Typically, the learning speed of a constant step-size SGD algorithm increases with the learning rate, up to a certain limit where the quality of the solution obtained decreases and the algorithm becomes unstable. By a "reasonable" learning rate value, we mean any value that is small enough to preserve the quality of the solution found and avoid instability.

7 We assume here that the pheromone variables are stored as a set of tables $\mathcal{T}_x = [\tau(x, y)]_{y \in X}$, each $\mathcal{T}_x$ being accessible to the artificial ants (or "physically located") in city $x \in X$.

where $\delta(x_{t+1} = x) = 1$ if $x_{t+1} = x$ and 0 otherwise, and $\Pr(x \mid \mathcal{T}, h_t)$ represents the probability that the ant chooses $x \in X$ as the next city when it stands in x_t and it has already followed the path $h_t \in \bar{X}^t$. Note that the ants need a little bit more memory than in the original AS. They need to remember not only the tour they followed, but also the probability of choosing each city at each step of the forward trip. If they do not have such a memory, they can always recompute the probabilities using the pheromone trails, but the trails must not have changed in between due to another ant updating pheromones. Therefore, this solution works exactly only in a synchronous implementation of the algorithm.

The previous results show that the SGD of the error $\mathrm{E}[1/L(h_n) \mid \mathcal{T}]$ is close to the original AS algorithm. This appears clearly when we compare Equation 7 to the analogous step-by-step update rule used in an asynchronous implementation of AS based on Equation 3:

$$\Delta\tau(x_t, x) = \rho\left(\frac{\delta(x_{t+1} = x)}{L(h_n)} - \tau(x_t, x)\right). \tag{8}$$

The step-size α of SGD plays the role of the evaporation rate ρ in AS. The main differences are[8]

- the pheromone value $\tau(x_t, x)$ in AS update rule is replaced in SGD by the probability $\Pr(x \mid \mathcal{T}, h_t)$ of moving from x_t to x;
- the decrease of pheromones (through the term $-\tau(x_t, x)$ in Equation 8, and through the term $-\Pr(x \mid \mathcal{T}, h_t)$ in Equation 7) is proportional to the *reward* $1/L(h_n)$ in our algorithm, whereas it is independent of it in the original AS;
- the presence of the factor $1/\tau(x_t, x)$ in the update rule (Equation 7) of the gradient-based algorithm.

It is important to note that, in our algorithm, the pheromones that are not used during an ant's forward trip are not modified during the ant's backward trip. If we had already visited y when we were in x, then $\tau(x, y)$ was not used to choose the next city after x, and hence, it was not used at all during the whole forward trip. As a consequence, $\tau(x, y)$ is left unchanged during the backward trip (that was not the case in the original AS where each pheromone trail evaporates). This makes sense, because if $\tau(x, y)$ is not used during the generation of a tour, then the value of this tour provides no information about the good way to change $\tau(x, y)$. This will be true in any application of SGD. It implies that the weight update associated with a forward trip (i.e., a sampled solution), can always be performed in a backward trip following the same path. This is the basis of "ant" implementations of SGD presented in this article.

There are a few problems with the algorithm we just defined. First, the update rule may bring the weights $\tau(x, y)$ at or below 0. Negative pheromone trails do not really make sense. Moreover, we supposed the pheromones to be (strictly) positive when calculating the gradient. When some pheromones are 0, the analytical expression of the gradient is more complex. An empirical solution to this problem consists of artificially preventing the weights from falling below a given value $\epsilon > 0$. However, there is another problem with this algorithm: the contribution $T_{x,y}(h_n)/L(h_n)$ of a sequence $h_n \in \bar{X}^n$ may tend to infinity when some pheromone trails $\tau(x, y)$ tend to 0, which induces a very unstable behavior of the algorithm in some regions of the search space.

8 Another difference between the two algorithms is that the pheromone $\tau(x_n, x_1)$ between the last and first city of the tour is reinforced by AS, whereas it is left untouched by SGD.

For instance, if $\tau(x, y) \approx 0$ for some $(x, y) \in \bar{X}^2$ and an ant unluckily goes through this edge during its forward trip, then the subsequent weight update, which is proportional to $1/\tau(x, y)$, may bring the algorithm very far from its original state. This is a case of unstable behavior due to unboundedness of the gradient estimate variance [2]: Although the expected value of the gradient estimate (i.e., the gradient itself) is always finite, the *variance* of the gradient estimate tends to infinity when some weights $\tau(x, y)$ tend to 0. In the next section, we present a new implementation of SGD that does not exhibit this instability and still belongs to the family of ACO algorithms.

2.4 A Stable ACO/SGD Algorithm

A classical solution to the problem of unbounded variance and unstable behavior is to use Boltzmann's law instead of the proportional selection rule of Equation 1 (e.g., [1]). In our case, the city-selection rule takes the form

$$\Pr(x_{t+1} = x \mid \mathcal{T}, h_t) = \begin{cases} 0 & \text{if } x \in h_t, \\ e^{\tau(x_t, x)} / \sum\limits_{\substack{y \in X \\ y \notin h_t}} e^{\tau(x_t, y)} & \text{otherwise.} \end{cases} \tag{9}$$

The derivation presented in Section 2.2 is still valid; the only changes are in the calculation of the traces (Sect. 2.3):

- if $x \neq x_{t-1}$ or $y \in h_{t-1}$, then

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} = 0;$$

- if $x = x_{t-1}$ and $y = x_t$ then

$$\begin{aligned} \frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} &= \frac{\partial \ln\left(e^{\tau(x,y)} / \sum\limits_{y' \notin h_{t-1}} e^{\tau(x,y')} \right)}{\partial \tau(x, y)}, \\ &= \frac{\partial \ln(e^{\tau(x,y)})}{\partial \tau(x, y)} - \frac{\partial \ln\left(\sum\limits_{y' \notin h_{t-1}} e^{\tau(x,y')} \right)}{\partial \tau(x, y)}, \\ &= 1 - \frac{e^{\tau(x,y)}}{\sum\limits_{y' \notin h_{t-1}} e^{\tau(x,y')}}, \\ &= 1 - \Pr(y \mid \mathcal{T}, h_{t-1}); \end{aligned}$$

- if $x = x_{t-1}$, $y \neq x_t$ and $y \notin h_{t-1}$ then, similarly,

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, h_{t-1}))}{\partial \tau(x, y)} = -\Pr(y \mid \mathcal{T}, h_{t-1}).$$

As in the previous case, the gradient-descent weight updates may be performed by the artificial ants during backward trips when they retrace their path backward. The new pheromone update rule, which replaces Equation 7, is

$$\Delta\tau(x_t, x) = \alpha \frac{1}{L(h_n)} (\delta(x_{t+1} = x) - \Pr(x \mid \mathcal{T}, h_t)). \tag{10}$$

Note that the factor $1/\tau(x_t, x)$ has disappeared from the right-hand side, making this rule very similar to the original AS update rule (Equation 8).

Ants need the same memory capacity as in the previous SGD algorithm. This new algorithm does not have the drawbacks of the previous one. The weights $\tau(x, y)$ are unconstrained and can take any real value while keeping the probability of each path (strictly) positive. Moreover, the gradient estimate is uniformly bounded (i.e., bounded by the same bound for all $h_n \in \bar{X}^n$) by the value $1/L^*$, where L^* is the length of the shortest path, and so its variance is bounded by $(1/L^*)^2$. Therefore, this algorithm is stable in any region of the search space.

Bounded variance is a necessary condition for convergence to a local optimum, but it is not sufficient [2]. As a matter of fact, it can be shown that stochastic gradient algorithms such as ours may exhibit unbounded behavior, that is, some weights may tend to infinity [28]. A typical case of unbounded behavior is when the weights that generate an optimal solution tend to plus infinity, while all the other weights tend to minus infinity. The problem is that an optimal set of pheromone values (i.e., a set that generates an optimal tour with probability 1) is obtained when some weights are positively or negatively infinite. Therefore, the algorithm may continue to climb a given choice forever, so that some weights diverge to infinity. However, this limitation has no consequence in our combinatorial optimization framework because we are not interested in having all the ants follow an optimal path; we just want an optimal solution to be generated (at least) once. Moreover, some classical tricks may be used to derive a formally convergent variant of our algorithm. Notably, we can

- either artificially bound the weights away from infinity (i.e., they are enforced to stay in a compact subset of $\mathbb{R}^l$, where l is the total number of weights), as in [28];
- or add to the objective function a penalty term in the form $-c \cdot \|\mathcal{T}\|$, where c is a constant and $\|\mathcal{T}\|$ is a norm of $\mathcal{T}$, so that the objective function tends to $-\infty$ when the weights tend to infinity [2].

We conjecture that either of these two solutions may be used to design a variant of our algorithm that converges with probability 1 to a local optimum of the error function.

2.5 Extensions

It is easy to modify the algorithm so that it optimizes other criteria than $\mathrm{E}[1/L(h_n) \mid \mathcal{T}]$. For instance, if we want to minimize the expected tour length $\mathrm{E}[L(h_n) \mid \mathcal{T}]$, then the update rule of the (stable) SGD algorithm becomes

$$\Delta\tau(x_t, x) = -\alpha L(h_n)(\delta(x_{t+1} = x) - \Pr(x \mid \mathcal{T}, h_t)).$$

Here, the algorithm may be understood as maximizing the reward $-L(h_n)$, which is always negative.[9] It is important to note that, because the shape of the objective function strongly determines the behavior of a gradient-following algorithm with constant step-size such as ours, the performance of SGD may vary with different objective functions, even if these functions have the same local and global optima.

More generally, the same approach could be applied to every combinatorial optimization problem for which an ACO algorithm can be designed.[10] The generic

9 Conversely to the original AS, our gradient algorithm does not assume that the objective function is positive.

10 That is, for which a constructive heuristic can be defined [9, 10].

ACO/SGD approach to a given maximization problem with solution set $\mathcal{S}$ and objective function $f\colon \mathcal{S} \to \mathbb{R}$ can be summarized as follows: First, design a stochastic controller that generates solutions in an iterative way using a set of weights $\mathcal{T}$. The controller is represented as a *construction graph* $\mathcal{G}$ such that the generation of a solution corresponds to some sort of path in this graph, and the weights $\mathcal{T}$ are attached to the arcs (or vertices) of $\mathcal{G}$. The weights, called pheromones, determine the transition probabilities in $\mathcal{G}$ during the random generation of a solution. This first stage, which is called the choice of a *problem representation* in [10], is crucial. It transforms the static combinatorial problem $\max\{f(s)\colon\ s \in \mathcal{S}\}$ into the problem of optimizing a trajectory, that is, a dynamic problem.

The next step is to define the "error" function as the expected value of a solution produced by the controller, given the current weights:

$$\mathcal{E} = \mathrm{E}[F(s) \mid \mathcal{T}] = \sum_{s\in\mathcal{S}} \Pr(s \mid \mathcal{T})F(s)$$

where $F\colon \mathcal{S} \to \mathbb{R}$ strictly increases with f (that is, $f(s) > f(s') \Longrightarrow F(s) > F(s')$).

Solutions s are generated by an iterative stochastic process that follows a finite number of steps. Let H be the set of trajectories in $\mathcal{G}$ that the controller may follow when generating a solution, and $g\colon H \to \mathcal{S}$ be the function that assigns the solution produced to a given trajectory. The error may be rewritten as:

$$\mathcal{E} = \mathrm{E}[F(g(h)) \mid \mathcal{T}] = \sum_{h\in H} \Pr(h \mid \mathcal{T})F(g(h)).$$

The gradient of the error is then the expectation over all possible trajectories of the value of the solution produced multiplied by an eligibility trace T_τ:

$$\frac{\partial \mathcal{E}}{\partial \tau} = \sum_{h\in H} \Pr(h \mid \mathcal{T})F(g(h))\,T_\tau(h) = \mathrm{E}[F(g(h))\,T_\tau(h) \mid \mathcal{T}],$$

for each individual weight τ. The trace $T_\tau(s)$ is the sum of the partial derivatives of the log of every elementary event that composes the trajectory s. Note that this decomposable structure of the gradient derives from the fact that solution generation is an iterative process, that is, from the very nature of the ACO approach.

Stochastic gradient descent can thus be implemented by sampling a few trajectories h—which can be seen as sending a few ants forward in the construction graph $\mathcal{G}$—and calculating their contribution to the gradient. In general, the weights that are not used when sampling a trajectory h have zero update according to the contribution of h. Therefore, the weight updates can be performed by artificial ants during backward trips in $\mathcal{G}$.[11] Finally, if the ants use Boltzmann's law to make random choices during forward trips, then the gradient estimate is uniformly bounded and the algorithm is stable in any part of the search space.

3 Discussion

Technically speaking, the ACO/SGD algorithm described above is not new. It is an instance of the generalized learning automaton [27] and, more precisely, of the gradient-

11 Note that the construction process does not necessarily have the same "taboo" aspect as in ATSP, where each weight is used at most once during an ant's forward trip. The calculation of the SGD update is not more difficult in this case. The general rule is that if an ant traverses a vertex of $\mathcal{G}$ (and uses the associated pheromone values) several times during its forward trip, it must traverse that vertex (and update the associated pheromone values) the same number of times during its backward trip.

based reinforcement learning algorithm known as REINFORCE [1, 35]. The originality of our work is to apply REINFORCE in the framework of ACO for combinatorial optimization, instead of the traditional Markov decision process (MDP) or partially observable MDP (POMDP) used in the reinforcement learning literature [20, 33]. Therefore, this work establishes connections between "classical" reinforcement learning and the less classical ACO learning. It suggests a general approach for applying reinforcement learning to combinatorial optimization that can be resumed as follows:

1. Design a parametrized stochastic controller that generates solutions *in an incremental way*, which turns the original static (optimization) problem into a dynamic (control) problem;
2. Use a reinforcement learning algorithm to (learn to) optimize the controller.

It is tempting to over-generalize the previous results and see in any ACO algorithm a more or less accurate approximation of the mechanism of gradient descent in the space of pheromones. In a sense, SGD is a very intuitive trial-and-error Monte-Carlo technique that samples solutions, increases the probability of the best sampled solutions, and decreases the probability of the worst. Its particularity is to be grounded on a solid theoretical framework so that it is possible to give a sense to the updates performed by the algorithm, but the basic intuition is the same as in ACO algorithms. Researchers in the field of ACO algorithms might find this position a little bit reductive. In fact, the best ACO algorithms are not limited to the simple trial-and-error ant search but also use other optimization techniques such as constructive heuristics and local search routines [16, 32]. However, these features may also be grounded in the gradient-descent framework and improve the algorithm's performance (see Sect. 3.1). It is also interesting to note that the assimilation of ACO to approximate SGD allows us to draw a parallel with artificial neural networks (ANNs), because SGD is the basic principle behind the well-known backpropagation algorithm [26, 30]. Accordingly, we suggest in this work that the basic mechanisms at work in ACO and ANNs could be the same. In the following, we discuss some opportunities of cross-fertilization between ACO and SGD and then survey some important issues in the study of ACO algorithms.

3.1 Mutual Contributions

There are many opportunities for cross-fertilization between ACO and SGD (or, more generally, gradient-based reinforcement learning). On one side, several efficient acceleration techniques for gradient-based reinforcement learning can easily be implemented in our ACO/SGD algorithm and then generalized to other ACO algorithms (this is the subject of ongoing research). On the other side, existing ACO algorithms suggest different ways of implementing the technique of SGD in the context of combinatorial optimization. The most successful applications of the metaheuristic are not limited to the simple trial-and-error ant search but also use some "external" information in the form of constructive heuristics or (discrete) local search routines. They suggest different ways of merging SGD and these two combinatorial optimization techniques. In this section, we examine how our simple ACO/SGD algorithm can be used in combination with these techniques, as inspired by previous ACO algorithms.

3.1.1 Using Constructive Heuristics

Many successful implementations of the ACO metaheuristic combine information from the pheromone trails and heuristic information when generating solutions. In the case of AS, the city selection rule (1) is replaced by

$$\Pr(x_{t+1} = x \mid \mathcal{T}, \eta, h_t) = \begin{cases} 0 & \text{if } x \in h_t, \\ \tau(x_t, x)^a \eta(x_t, x)^b / \sum_{\substack{y \in X \\ y \notin h_t}} \tau(x_t, y)^a \eta(x_t, y)^b & \text{otherwise,} \end{cases}$$

where $\eta \geq 0$ is a (constructive) heuristic function of $(x, y) \in \bar{X}^2$, and a and b are two (positive) parameters that determine the relative influence of pheromone trails and heuristic information. The function η reflects heuristic information about the good way to complete a partial solution. For instance, in the case of ATSP, a common choice is $\eta(x, y) = 1/d(x, y)$ for all $(x, y) \in \bar{X}^2$. In this case, the closest unvisited cities have larger probability to be chosen than without heuristic information. Moreover, in the successful applications of ACO to *nonstationary* (time-varying) problems, such as data packet routing in AntNet [6], the function η is used to provide information to the algorithm about the current state of the problem.

There are several ways of integrating a similar mechanism in our algorithm. A particularly simple and elegant formulation is obtained when we replace the exponential selection rule (9) of our gradient algorithm with the following equation:

$$\Pr(x_{t+1} = x \mid \mathcal{T}, \eta, h_t) = \begin{cases} 0 & \text{if } x \in h_t, \\ e^{a\tau(x_t, x) + b\eta(x_t, x)} / \sum_{\substack{y \in X \\ y \notin h_t}} e^{a\tau(x_t, y) + b\eta(x_t, y)} & \text{otherwise,} \end{cases} \quad (11)$$

where a and $b \geq 0$ are two external parameters that play the same role as in the previous equation.[12] Equation 9 is obtained when $a = 1$ and $b = 0$. When $a = 0$ and $b > 0$, the algorithm does not use the pheromone trails at all. It is then an iterative stochastic heuristic search similar to the first stage of GRASP [15].

Therefore, we dispose of a whole range of algorithms that extend from pure (heuristic-free) gradient-based reinforcement learning to simple (constant probability) stochastic heuristic search.

The next step is to recalculate the gradient to take into account the new selection rule. Once again, only the last part of the calculation is modified. We have the following:

- if $x \neq x_{t-1}$ or $y \in h_{t-1}$, then

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, \eta, h_{t-1}))}{\partial \tau(x, y)} = 0;$$

- if $x = x_{t-1}$ and $y = x_t$ then

$$\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, \eta, h_{t-1}))}{\partial \tau(x, y)} = \frac{\partial \ln\left(e^{a\tau(x,y) + b\eta(x,y)} / \sum_{y' \notin h_{t-1}} e^{a\tau(x,y') + b\eta(x,y')}\right)}{\partial \tau(x, y)},$$

12 Note that this formalism is compatible with negative heuristic functions. Therefore, we could choose $\eta(x, y) = -d(x, y)$ in the case of ATSP.

$$
\begin{aligned}
&= \frac{\partial \ln(\mathrm{e}^{a\tau(x,y)+b\eta(x,y)})}{\partial \tau(x,y)} - \frac{\partial \ln\left(\sum_{y' \notin h_{t-1}} \mathrm{e}^{a\tau(x,y')+b\eta(x,y')}\right)}{\partial \tau(x,y)},\\
&= a\left(1 - \frac{\mathrm{e}^{a\tau(x,y)+b\eta(x,y)}}{\sum_{y' \notin h_{t-1}} \mathrm{e}^{a\tau(x,y')+b\eta(x,y')}}\right),\\
&= a(1 - \Pr(y \mid \mathcal{T}, \eta, h_{t-1}));
\end{aligned}
$$

- if $x = x_{t-1}$, $y \neq x_t$, and $y \notin h_{t-1}$ then, similarly,

$$
\frac{\partial \ln(\Pr(x_t \mid \mathcal{T}, \eta, h_{t-1}))}{\partial \tau(x,y)} = -a\Pr(y \mid \mathcal{T}, \eta, h_{t-1}).
$$

This leads to the following pheromone update rule, after absorbing the constant factor a in the learning rate α:

$$
\Delta\tau(x_t, x) = \alpha \frac{1}{L(h_n)} (\delta(x_{t+1} = x) - \Pr(x \mid \mathcal{T}, \eta, h_t))
$$

The only difference with the update rule of the previous algorithm (Equation 10) is that the heuristic-independent probability $\Pr(x \mid \mathcal{T}, h_t)$ calculated following Equation 9 is replaced by the heuristic-dependent probability $\Pr(x \mid \mathcal{T}, \eta, h_t)$ defined by Equation 11. Therefore, the basic principle of the heuristic-free algorithm generalizes to the new selection rule: Each ant has to memorize the probability distribution it uses at each step of its forward trip and then decrease the pheromones on its way back proportionally to these distributions.

It is important to note that, by varying the values of the external parameters a and b, we change two factors that strongly influence the effectiveness of the algorithm. The first is the shape of the "error" function $\mathcal{E}$ (defined as a function from pheromone vectors to real number) that the algorithm is approximately "descending." It is not clear to us what is exactly the effect of the new selection rule on the "landscape" we are exploring. A globally optimum set of pheromones is still obtained by putting infinite weights on the best paths (that do not change when changing the parameter values), but some important aspects, such as the steepness of some peaks, are modified. The second and probably most important feature that is modified by varying parameters a and b is the sampling process used to estimate the gradient. The probability of sampling different paths and the update of the corresponding pheromones changes as a function of the parameter values. In the case of ATSP, the overall effect of using heuristic information (i.e., having $b > 0$) is that when an unvisited city y is close to the current city x, it has a greater chance to be chosen than without heuristic information. On the other side, the pheromone trail $\tau(x, y)$ is decreased by a larger amount each time y is considered as a candidate successor of x. Further research is needed to understand how better performance may be obtained using heuristic information appropriately.

3.1.2 Using Discrete Local Search

The ACO metaheuristic is often used in conjunction with local search algorithms [16, 32]. In this approach, an ACO algorithm generates starting points for a discrete local search routine.[13] Each ant produces a solution, say s_1, which is then transformed into another

13 Gradient descent is itself a local search procedure, but it operates in the continuous space of pheromones, whereas the discrete local search used here operates in the discrete set of solutions.

solution, say s_2, by the local search. Then the pheromones are updated. As our goal is to maximize the quality of the final solution s_2, pheromone updates must be proportional to the quality of s_2, not s_1. Given this, there are still two ways of updating the pheromones:

- either we reinforce the pheromones corresponding to the final solution s_2—in other words, we do as if the solution s_2 was generated directly by the ant algorithm, without the help of the local search (in this approach, we suppose that there is a mapping between the solution set and the set of possible forward trajectories);
- or we reinforce the pheromones corresponding to the intermediate solution s_1.

By analogy with similar procedures in the area of genetic algorithms [34], we call the first alternative the Lamarckian approach, and the second the Darwinian approach.

The main argument supporting the Lamarckian approach is that it is reasonable to think that, if the ant algorithm can be trained directly using the better solution s_2, then it would be stupid to train it using the worse solution s_1. In fact, in published ACO implementations, only the Lamarckian alternative has been used. In the case of SGD, however, the Darwinian approach may make more sense. It is easy to show that, if we try to maximize the expected value of the solution s_2 produced by the local search algorithm, then the update rule of an SGD algorithm is to reinforce the pheromones corresponding to the intermediate solution s_1 proportionally to the value of the final solution s_2. The formal framework developed in Section 2.5 can be used for this calculation, the effect of the local search being modeled in the function F. Having understood this, we can derive qualitative arguments in favor of the Darwinian approach. For instance, if the good starting points of the local search are very far from the corresponding local optima in the topology of the gradient algorithm, then the Darwinian approach could outperform the Lamarckian.

3.2 Important Issues

By establishing connections with ANNs on one side, and reinforcement learning on the other side, we also show that ACO algorithms are concerned with two important issues paradigmatically illustrated in these techniques. They are, respectively, the issue of *generalization* and the *exploration* versus *exploitation dilemma*. In this section, we examine how these problems arise in ACO algorithms. It is clear that *any* reinforcement learning algorithm for combinatorial optimization has to deal with these two issues simultaneously.

3.2.1 Generalization

The most famous application of SGD is surely the algorithm known as backpropagation in ANNs, and the issue the most studied in backpropagation is probably the ability of the algorithm to generalize over inputs [30]. Stated simply, the problem is to find a set of weights $W = [w]$ for a network encoding a function F from an input set I to an output set O (say, $O = \mathbb{R}$), so that F approximates as much as possible a target function F^*: $I \to O$. Backpropagation learns an optimal configuration of weights by observing a set of training examples, that is, pairs $(i, F^*(i))$ with $i \in I$, and memorizing and generalizing these observations. In general, the input set I is a huge combinatorial set, if not an infinite set. Therefore, it is not possible to present every instance $i \in I$ in the training set. However, backpropagation is able to generalize the observed data to unseen instances. That is, it assumes that any unseen input i has a value $F^*(i)$ that is close to the value of the observed examples that are similar to i in some sense. It is well known that the ability of an ANN to generalize and its efficiency in generalization strongly depend on the network structure [5].

As we said, the backpropagation algorithm is an instance of SGD. More precisely, its overall effect is to descend the gradient of the mean square error

$$\mathcal{E}_{\mathrm{MS}} = \sum_{i \in I} p_i (F(i) - F^*(i))^2 = \mathrm{E}[(F - F^*)^2 \mid W],$$

where $[p_i]_{i \in I}$ is a given probability distribution on instances ($\sum_{i \in I} p_i = 1$). Note that, differently from the case of our ACO/SGD algorithm, the expectation in this equation is conditional on the weights W because the local error

$$\mathrm{e}_{\mathrm{MS}} \overset{\mathrm{def}}{=} (F - F^*)^2$$

depends on the weights, whereas the probabilities p_i do not. Therefore,

$$\frac{\partial \mathcal{E}_{\mathrm{MS}}}{\partial w} = \sum_{i \in I} p_i \frac{\partial \mathrm{e}_{\mathrm{MS}}(i)}{\partial w} = \mathrm{E}\left[\frac{\partial \mathrm{e}_{\mathrm{MS}}}{\partial w} \mid W \right].$$

This result suggests an exact gradient algorithm that enumerates all possible inputs $i \in I$ for each step of gradient descent. Conversely, backpropagation uses an unbiased estimate of the gradient obtained by sampling a unique input $i \in I$. After sampling input i and comparing the actual output $F(i)$ and the desired output $F^*(i)$, backpropagation updates the weights of the network following

$$\Delta w = -\alpha \frac{\partial \mathrm{e}_{\mathrm{MS}}(i)}{\partial w}.$$

for each weight w. The values of $F(i)$ and $F^*(i)$ are used here to calculate the partial derivative $\partial \mathrm{e}_{\mathrm{MS}}(i)/\partial w$.

We see that the basic principles of backpropagation and of our algorithm are the same. It is, in both cases, a Monte-Carlo estimation of the gradient of a given error function, with respect to a set of weights attached to the components of a graph (the construction graph $\mathcal{G}$ in one case, and the ANN itself in the other case). Also, both algorithms are distributed and parallel implementation of this principle. Therefore, our algorithm should have, at least partially, the same ability to generalize observed data over unseen instances as backpropagation.

Actually, it is not difficult to convince oneself that generalization is as big an issue in ACO in general as in ANNs. For instance, it is clear that the application of AS to the ATSP works by generalizing the observed solutions: If a majority of the sampled tours that traverse a given arc $(x, y) \in \bar{X}^2$ are of good quality, then the algorithm increases the probability of traversing this arc. In a sense, it assumes that, *in general*, the tours that traverse (x, y) are of good quality. That is, it generalizes the observations. As in the case of backpropagation, the ability and efficiency of an ACO algorithm to generalize is mostly determined by the structure of the graph, that is, the problem representation used by the artificial ants.

3.2.2 Exploration versus Exploitation

As we stressed above, the ACO metaheuristic can be seen as a way of applying reinforcement learning to combinatorial optimization problems. Thus, every ACO algorithm has to deal with one of the main issues in reinforcement learning: the exploration versus exploitation dilemma [19]. This is the problem of finding an optimal compromise between obtaining additional information about the least known solutions (exploration), and maximizing rewards by taking the estimated best actions (exploitation). This problem is characteristic of real-time on-line learning, where one motivation is to learn as fast as possible, and another is to maximize the reward received during each experience.

In the case of ACO algorithms, the mechanisms of the exploration versus exploitation dilemma are intimately linked with those of generalization. For instance, in the application to the ATSP, since the pheromones are attached to pairs of cities $(x, y) \in \bar{X}^2$, the algorithm is confronted with questions like: "Is it necessary to try again a given pair of cities that seems to be nonoptimal?" Clearly, different questions arise with different problem representations. In fact, it appears that an optimal solution to the exploration versus exploitation dilemma in the framework of ACO depends intimately on the problem representation, that is, the structure of the construction graph.

In practice, our ACO/SGD algorithm does not really address the issue of exploration, although it does face the dilemma. Its behavior is dictated by the trial-and-error search of SGD, independent of any consideration about exploration.

4 Conclusions

In this article, we explored the connections between the two techniques of ACO and SGD. First, we showed that the empirical designed AS algorithm is very similar to SGD in the space of pheromones, and we proposed a stable implementation of gradient-based reinforcement learning that belongs to the framework of ACO algorithms. Then we outlined a general ACO/SGD algorithm for combinatorial optimization. The performance of this algorithm depends crucially on some basic choices such as the problem representation and the objective function. This insight may be used to develop simple acceleration techniques for ACO algorithms, by transposing previous work on gradient-based reinforcement learning. Moreover, the most successful applications of the ACO metaheuristic suggest new ways of merging gradient descent with other optimization techniques for combinatorial optimization.

In conclusion, we believe that this work constitutes a significant step toward understanding the mechanisms at work in ACO algorithms, shedding new light on some important issues in the theory of these algorithms.

Acknowledgments
This work was supported by a Marie Curie Fellowship awarded to Nicolas Meuleau (Contract No. HPMFCT-2000-00230). Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate. This work was also partially supported by the "Metaheuristics Network," a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Baird, L. C., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. In M. Kearns, S. Solla, & D. Cohn (Eds.), *Advances in neural information processing systems, 11* (pp. 968–974). Cambridge, MA: MIT Press.
2. Bertsekas, D. P. (1995). *Nonlinear programming*. Belmont, MA: Athena Scientific.
3. Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. New York: Oxford University Press.
4. Bonabeau, E., Dorigo, M., & Theraulaz, G. (2000). Inspiration for optimization from social insect behavior. *Nature, 406*, 39–42.

5. Boyanov, K. (Ed.). (1990). *Parallel and distributed processing*. Amsterdam: North-Holland.
6. Di Caro, G., & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research, 9*, 317–365.
7. Dorigo, M. (1992). *Optimization, learning and natural algorithms* (in Italian). Unpublished doctoral dissertation, Dipartimento di Elettronica, Politecnico di Milano, Italy.
8. Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems, 16*(8), 851–871.
9. Dorigo, M., & Di Caro, G. (1999). The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New ideas in optimization* (pp. 11–32). London: McGraw-Hill.
10. Dorigo, M., Di Caro, G., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life, 5*, 137–172.
11. Dorigo, M., Di Caro, G., & Stützle, T. (Eds.). (2000). Special issue on "ant algorithms". *Future Generation Computer Systems, 16*(8), 851–956.
12. Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.
13. Dorigo, M., Gambardella, L. M., Middendorf, M., & Stützle, T. (Eds.). (in press). Ant algorithms and Swarm intelligence [Special issue]. *IEEE Transactions on Evolutionary Computation, 6*(4).
14. Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, 26*(1), 29–41.
15. Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8*, 67–71.
16. Gambardella, L. M., & Dorigo, M. (2000). Ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing, 12*(3), 237–255.
17. Gambardella, L. M., Taillard, È. D., & Agazzi, G. (1999). MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New ideas in optimization* (pp. 63–76). London: McGraw-Hill.
18. Gambardella, L. M., Taillard, È. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society, 50*(2), 167–176.
19. Kaelbling, L. P. (1993). *Learning in embedded systems*. Cambridge, MA: MIT Press.
20. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.
21. Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The travelling salesman problem*. Chichester, UK: Wiley.
22. Maniezzo, V., & Carbonaro, A. (2000). An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems, 16*(8), 927–935.
23. Maniezzo, V., & Colorni, A. (1999). The ant system applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering, 11*(5), 769–778.
24. Merkle, D., Middendorf, M., & Schmeck, H. (2000). Ant colony optimization for resource-constrained project scheduling. In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, & H.-G. Beyer (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (pp. 893–900). San Francisco: Morgan Kaufmann.
25. Michel, R., & Middendorf, M. (1999). An ACO algorithm for the shortest supersequence problem. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New ideas in optimization* (pp. 51–61). London: McGraw-Hill.
26. Mitchell, T. (1997). *Machine learning*. Boston: McGraw-Hill.

27. Narendra, K. S., & Thathachar, M. A. L. (1989). *Learning automata: An introduction.* Englewood Cliffs, NJ: Prentice-Hall.

28. Phansalkar, V. V., & Thathachar, M. A. L. (1995). Local and global optimization algorithms for generalized learning automata. *Neural Computation, 7*, 950–973.

29. Robbins, H., & Monroe, H. (1951). A stochastic approximation method. *Annals of Mathematics and Statistics, 22*, 400–407.

30. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error backpropagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructures of cognition: Vol. 1. Foundations* (pp. 318–362). Cambridge, MA: MIT Press.

31. Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. *Adaptive Behavior, 5*(2), 169–207.

32. Stützle, T., & Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)* (pp. 309–314). Piscataway, NJ: IEEE Press.

33. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

34. Whitley, D., Gordon, S., & Mathias, K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. In *Proceedings of PPSN-III, Third International Conference on Parallel Problem Solving from Nature* (pp. 6–15). Berlin: Springer.

35. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256.