Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 185 (2008) 1155–1173
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Ant colony optimization for continuous domains

Krzysztof Socha *, Marco Dorigo

IRIDIA, Université Libre de Bruxelles, CP 194/6, Ave. Franklin D. Roosevelt 50, 1050 Brussels, Belgium[1]

Received 1 August 2005; accepted 1 June 2006
Available online 3 November 2006

Abstract

In this paper we present an extension of ant colony optimization (ACO) to continuous domains. We show how ACO, which was initially developed to be a metaheuristic for combinatorial optimization, can be adapted to continuous optimization without any major conceptual change to its structure. We present the general idea, implementation, and results obtained. We compare the results with those reported in the literature for other continuous optimization methods: other ant-related approaches and other metaheuristics initially developed for combinatorial optimization and later adapted to handle the continuous case. We discuss how our extended ACO compares to those algorithms, and we present some analysis of its efficiency and robustness.

Keywords: Ant colony optimization; Continuous optimization; Metaheuristics

1. Introduction

Optimization algorithms inspired by the ants' foraging behavior (Dorigo, 1992) have been initially proposed for solving combinatorial optimization problems (COPs). Examples of COPs include scheduling, vehicle routing, timetabling, and so on. Many of these problems, especially those of practical relevance, are NP-hard. In other words, it is strongly believed that it is not possible to find efficient (i.e., polynomial time) algorithms to solve them optimally. Usually these problems are tackled with heuristic methods (i.e., not exact methods) that permit to find approximate solutions (i.e., solutions that are good, but not provably optimal) in a reasonable amount of time.

Combinatorial optimization—as the name suggests—deals with finding optimal *combinations* or *permutations* of available problem components. Hence, it is required that the problem is partitioned into a finite set of components, and the combinatorial optimization algorithm attempts to find their optimal combination or permutation. Many real-world optimization problems may be represented as COPs in a straightforward way. There exists however an important class of problems for which this is not the case: the class of optimization problems that require choosing values for continuous variables. Such problems may be tackled with a combinatorial optimization algorithm only once the continuous *ranges* of allowed values are converted into finite

* Corresponding author.
E-mail addresses: ksocha@ulb.ac.be (K. Socha), mdorigo@ulb.ac.be (M. Dorigo).
[1] http://iridia.ulb.ac.be

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.06.046

sets. This is not always convenient, especially if the initial possible range is wide, and the resolution required is very high. In these cases, algorithms that can natively handle continuous variables usually perform better. This paper presents a way to effectively apply Ant Colony Optimization (ACO)—an algorithm originally developed to tackle COPs—to continuous optimization problems.

Since the emergence of ACO as a combinatorial optimization tool, attempts have been made to use it for tackling continuous problems. However, applying the ACO metaheuristic to continuous domains was not straightforward, and the methods proposed often took inspiration from ACO, but did not follow it exactly.

Contrary to those earlier approaches, this paper presents a way to extend ACO to continuous domains without the need to make any major conceptual change to its structure. To improve the clarity of the paper, we denote this ACO extended to continuous domains by $\mathrm{ACO}_{\mathbb{R}}$. We aim at presenting the core idea of applying $\mathrm{ACO}_{\mathbb{R}}$ to continuous domains as well as an implementation that performs well on standard benchmark test problems.

Continuous optimization is hardly a new research field. There exist numerous algorithms—including metaheuristics—that were developed for tackling this type of problems. In order to have a proper perspective on the performance of $\mathrm{ACO}_{\mathbb{R}}$, we compare it not only to other ant-related methods, but also to other metaheuristics used for continuous optimization.

It is worth mentioning that $\mathrm{ACO}_{\mathbb{R}}$, due to its closeness to the original formulation of ACO, provides an additional advantage—the possibility of tackling mixed discrete–continuous optimization problems. In other words, with $\mathrm{ACO}_{\mathbb{R}}$ it should now be possible to consider problems where some variables are discrete and others are continuous. This possibility is however not explored in this paper—it is the subject of ongoing research. Here, we focus on tackling purely continuous optimization problems.

The reminder of the paper is organized as follows. Section 2 briefly overviews ACO. Section 3 presents $\mathrm{ACO}_{\mathbb{R}}$, our extension of ACO to tackle continuous optimization problems. Section 4 provides a discussion of the proposed approach with regard to other methods for continuous optimization. Section 5 presents the experimental setup and results obtained, and compares them to the results found in the literature. Finally, Section 6 presents the conclusions and future work plans. Additionally, Appendix A discusses an important issue concerning variable correlation handling in $\mathrm{ACO}_{\mathbb{R}}$.

2. Ant Colony Optimization

In the 1990's, Ant Colony Optimization was introduced as a novel nature-inspired method for the solution of hard combinatorial optimization problems (Dorigo, 1992; Dorigo et al., 1996, 1999; Dorigo and Stützle, 2004). The inspiring source of ACO is the foraging behavior of real ants. When searching for food, ants initially explore the area surrounding their nest in a random manner. As soon as an ant finds a food source, it evaluates it and carries some food back to the nest. During the return trip, the ant deposits a pheromone trail on the ground. The pheromone deposited, the amount of which may depend on the quantity and quality of the food, guides other ants to the food source. As it has been shown (Goss et al., 1989), indirect communication among ants via pheromone trails enables them to find shortest paths between their nest and food sources. This capability of real ant colonies has inspired the definition of artificial ant colonies that can find approximate solutions to hard combinatorial optimization problems.

The central component of ACO algorithms is the pheromone model, which is used to probabilistically sample the search space. As shown in Blum (2004), it can be derived from the model of the COP under consideration. A model of a COP may be defined as follows:

Definition 2.1. A model $P = (\mathbf{S}, \Omega, f)$ of a COP consists of:

- a search space $\mathbf{S}$ defined over a finite set of discrete decision variables and a set Ω of constraints among the variables;
- an objective function $f : \mathbf{S} \to \mathbb{R}_0^+$ to be minimized.

The search space $\mathbf{S}$ is defined as follows: Given is a set of discrete variables X_i, $i = 1, \ldots, n$, with values $v_i^j \in \mathbf{D}_i = \left\{ v_i^1, \ldots, v_i^{|\mathbf{D}_i|} \right\}$. A variable instantiation, that is, the assignment of a value v_i^j to a variable X_i, is denoted by $X_i \leftarrow v_i^j$. A solution $s \in \mathbf{S}$—i.e., a complete assignment, in which each decision variable has a value assigned—that satisfies all the constraints in the set Ω, is a feasible solution of the given COP. If the set Ω is empty, P is called an

unconstrained problem model, otherwise it is called a constrained one. A solution $s^* \in \mathbf{S}$ is called a global optimum if and only if: $f(s^*) \leqslant f(s) \quad \forall s \in \mathbf{S}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^* \subseteq \mathbf{S}$. Solving a COP requires finding at least one $s^* \in \mathbf{S}^*$.

The model of a COP is used to derive the pheromone model used by ACO. First, an instantiated decision variable $X_i = v_i^j$ (i.e, a variable X_i with a value v_i^j assigned from its domain $\mathbf{D}_i$), is called a *solution component* and denoted by c_{ij}. The set of all possible solution components is denoted by $\mathbf{C}$. A pheromone trail parameter T_{ij} is then associated with each component c_{ij}. The set of all pheromone trail parameters is denoted by $\mathbf{T}$. The value of a pheromone trail parameter T_{ij} is denoted by τ_{ij} (and called pheromone value).[2] This pheromone value is then used and updated by the ACO algorithm during the search. It allows modeling the probability distribution of different components of the solution.

2.1. The ACO metaheuristic

The ACO metaheuristic is shown in Algorithm 1. It consists of three algorithmic components that are gathered in the *ScheduleActivities* construct. The ScheduleActivities construct does not specify how these three activities are scheduled and synchronized, a decision that is left to the algorithm designer. In the following, we explain these three algorithmic blocks in more detail.

Algorithm 1

Ant Colony Optimization metaheuristic

```
while termination conditions not met do
   ScheduleActivities
      AntBasedSolutionConstruction( )
      PheromoneUpdate( )
      DaemonActions( )      {optional}
   end ScheduleActivities
endwhile
```

AntBasedSolutionConstruction( ): Artificial ants construct solutions from sequences of solution components taken from a finite set of n available solution components $\mathbf{C} = \{c_{ij}\}$. A solution construction starts with an empty partial solution $s^p = \emptyset$. Then, at each construction step, the current partial solution s^p is extended by adding a feasible solution component from the set $N(s^p) \in \mathbf{C} \backslash s^p$, which is defined by the solution construction mechanism. The process of constructing solutions can be regarded as a path on the construction graph $G_C = (\mathbf{V}, \mathbf{E})$. The set of solution components $\mathbf{C}$ may be associated either with the set $\mathbf{V}$ of vertices of the graph G_C, or with the set $\mathbf{E}$ of its edges. The allowed paths in G_C are implicitly defined by the solution construction mechanism that defines the set $N(s^p)$ with respect to a partial solution s^p.

The choice of a solution component from $N(s^p)$ is done probabilistically at each construction step. The exact rules for probabilistic choice of solution components vary across different variants of ACO. The basic and the best known is that of Ant System (AS) (Dorigo et al., 1996):

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{il}^{\alpha} \cdot \eta(c_{il})^{\beta}}, \quad \forall c_{ij} \in N(s^p), \qquad (1)$$

where τ_{ij} is the pheromone value associated with component c_{ij}, and $\eta(\cdot)$ is a weighting function that assigns at each construction step a heuristic value to each feasible solution component $c_{ij} \in N(s^p)$. The values that are given by the weighting function are commonly called the heuristic information. Furthermore, α and β are positive parameters, whose values determine the relation between pheromone information and heuristic information.

PheromoneUpdate( ): The aim of pheromone update is to increase the pheromone values associated with good or promising solutions, and decrease those that are associated with bad ones. Usually, this is achieved by increasing the pheromone levels associated with chosen good solution s_{ch} by a certain value $\Delta\tau$, and by decreasing all the pheromone values through *pheromone evaporation*:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau & \text{if} \quad \tau_{ij} \in s_{ch}, \\ (1-\rho)\tau_{ij} & \text{otherwise}, \end{cases} \qquad (2)$$

where $\rho \in (0, 1]$ is the evaporation rate. Pheromone evaporation is needed to avoid too rapid convergence of the algorithm. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space.

In general, good solutions found earlier by the ants are used to update the pheromone in order to increase the probability of the search by subsequent ants in the promising regions of the search space. Different ACO algorithms, such as for example

[2] Note that pheromone values are in general a function of the algorithm's iteration t: $\tau_{ij} = \tau_{ij}(t)$.

Ant Colony System (ACS) (Dorigo and Gambardella, 1997) or $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle and Hoos, 2000) differ in the way they update the pheromone. In principle, algorithms update pheromone using either the *iteration-best solution*—i.e., the best solution found in the last iteration, or the *best-so-far solution*—i.e., the best solution found from the start of the algorithm run (sometimes a combination of several solutions found by the ants is used). The best-so-far solution update leads to a faster convergence, while the iteration-best update allows for more diversification of the search (Stützle and Dorigo, 1999).

DaemonActions( ): Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples include the application of local search to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective.

2.2. ACO applications

After the initial proof-of-concept application to the traveling salesman problem (TSP) (Dorigo, 1992; Dorigo et al., 1996), ACO was applied to many other COPs. Examples include the applications to assignment problems (Costa and Hertz, 1997; Stützle and Hoos, 2000; Socha et al., 2003), scheduling problems (Merkle et al., 2002; Gagné et al., 2002; Blum and Sampels, 2004), and vehicle routing problems (Gambardella et al., 1999; Reimann et al., 2004).

3. ACO for continuous domain—$\mathrm{ACO}_{\mathbb{R}}$

Similarly to a combinatorial optimization problem (COP), also a model for continuous optimization problem (CnOP) may be formally defined:

Definition 3.1. A model $Q = (\mathbf{S}, \Omega, f)$ of a CnOP consists of:

- a search space $\mathbf{S}$ defined over a finite set of continuous decision variables and a set Ω of constraints among the variables;
- an objective function $f : \mathbf{S} \to \mathbb{R}_0^+$ to be minimized.

The search space $\mathbf{S}$ is defined as follows: Given is a set of continuous variables X_i, $i = 1, \ldots, n$, with values $v_i \in \mathbf{D}_i \subseteq \mathbb{R}$. A variable instantiation, that is, the assignment of a value v_i to a variable X_i, is denoted by $X_i \leftarrow v_i$. A solution $s \in \mathbf{S}$—i.e., a complete assignment, in which each decision variable has a value assigned—that satisfies all the constraints in the set Ω, is a feasible solution of the given CnOP. If the set Ω is empty, Q is called an unconstrained problem model, otherwise is called a constrained one. A solution $s^* \in \mathbf{S}$ is called a global optimum if and only if: $f(s^*) \leqslant f(s) \quad \forall s \in \mathbf{S}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^* \subseteq \mathbf{S}$. Solving a CnOP requires finding at least one $s^* \in \mathbf{S}^*$.

The idea that is central to the way ACO works is the incremental construction of solutions based on the biased (by pheromone) probabilistic choice of solution components. In ACO applied to combinatorial optimization problems, the set of available solution components is defined by the problem formulation. At each construction step, ants make a probabilistic choice of the solution component c_i from the set $N(s^p)$ of available components according to Eq. (1). The probabilities associated with the elements of the set $N(s^p)$ make up a *discrete probability distribution* (Fig. 1(a)) that an ant samples in order to choose a component to be added to the current partial solution s^p.

The fundamental idea underlying $\mathrm{ACO}_{\mathbb{R}}$ is the shift from using a *discrete* probability distribution to using a *continuous* one, that is, a *probability density function* (PDF) (Fig. 1(b)). In $\mathrm{ACO}_{\mathbb{R}}$, instead of choosing a component $c_{ij} \in N(s^p)$ according to Eq. (1), an ant samples a PDF. In the following sections we explain how this is accomplished. In Section 3.1, we present briefly the idea of using a PDF, and in Section 3.2 we outline the pheromone representation used in $\mathrm{ACO}_{\mathbb{R}}$. Finally, in Section 3.3 we give a detailed description of the $\mathrm{ACO}_{\mathbb{R}}$ algorithm itself. Additionally, Appendix A presents some specific aspects concerning variable correlation which are explicitly handled by $\mathrm{ACO}_{\mathbb{R}}$.

3.1. Probability density function (PDF)

Before going into in-depth description of the $\mathrm{ACO}_{\mathbb{R}}$ algorithm, we must discuss certain characteristics of PDFs, and select the one that we will use. In principle, a *probability density function* may be any function $P(x) \geqslant 0 \; \forall x$ such that:

$$\int_{-\infty}^{\infty} P(x)\mathrm{d}x = 1. \tag{3}$$

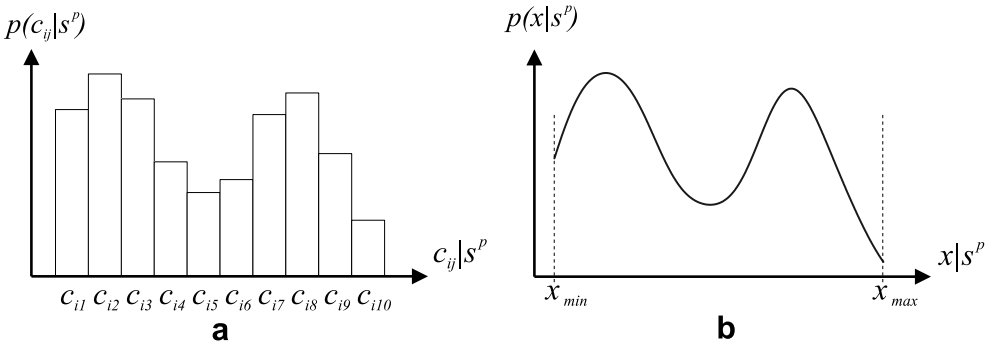

Fig. 1. (a) Discrete probability distribution $P_d(c_{ij}|s^p)$ of a finite set $\{c_{i1},\ldots,c_{i10}\} \in N(s^p)$ of available components. (b) Continuous probability density function $P_c(x|\ s^p)$ with possible range $x \in [x_{\min}, x_{\max}]$. The y-axis on both plots indicates the probability p. Note that $\sum_{j=1}^{10} p(c_{ij}|s^p) = \int_{x_{\min}}^{x_{\max}} p(x|s^p)\mathrm{d}x = 1$.

For a given probability density function $P(x)$, an associated *cumulative distribution function* (CDF) $D(x)$ may be defined, which is often useful when sampling the corresponding PDF. The CDF $D(x)$ associated with PDF $P(x)$ is defined as follows:

$$D(x) = \int_{-\infty}^{x} P(t)\mathrm{d}t. \tag{4}$$

The general approach to sampling PDF $P(x)$ is to use the *inverse* of its CDF, $D^{-1}(x)$. When using the inverse of the CDF, it is sufficient to have a pseudo-random number generator that produces uniformly distributed real numbers.[3] However, it is important to note that for an arbitrarily chosen PDF $P(x)$, it is not always straightforward to find $D^{-1}(x)$.

One of the most popular functions that is used as a PDF is the Gaussian function. It has some clear advantages, such as a reasonably easy way of sampling—e.g., the Box–Muller method (Box and Muller, 1958)—but it also has some disadvantages. A single Gaussian function is not able to describe a situation where two disjoint areas of the search space are promising, as it only has one maximum. Due to this fact, we use a PDF based on Gaussian functions, but slightly enhanced—a Gaussian kernel PDF. Similar constructs have been used before (Bosman and Thierens, 2002), but not exactly in the same way. We define a Gaussian kernel as a weighted sum of several one-dimensional Gaussian functions $g_l^i(x)$, and denote it as $G^i(x)$:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i\sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i\,2}}}. \tag{5}$$

Since we use as many Gaussian kernel PDFs as the number of dimensions of the problem, $i = 1,\ldots,n$ identifies a single such PDF. The Gaussian kernel $G^i(x)$ is parameterized with three vectors of parameters: $\boldsymbol{\omega}$ is the vector of weights associated with the individual Gaussian functions, $\boldsymbol{\mu}^i$ is the vector of means, and $\boldsymbol{\sigma}^i$ is the vector of standard deviations. The cardinality of all these vectors is equal to the number of Gaussian functions constituting the Gaussian kernel. For convenience, we will use the parameter k to describe it, hence $|\boldsymbol{\omega}| = |\boldsymbol{\mu}^i| = |\boldsymbol{\sigma}^i| = k$.

Such a PDF allows a reasonably easy sampling, and yet provides a much increased flexibility in the possible shape, in comparison to a single Gaussian function. An example of how such a Gaussian kernel PDF may look like is presented in Fig. 2.

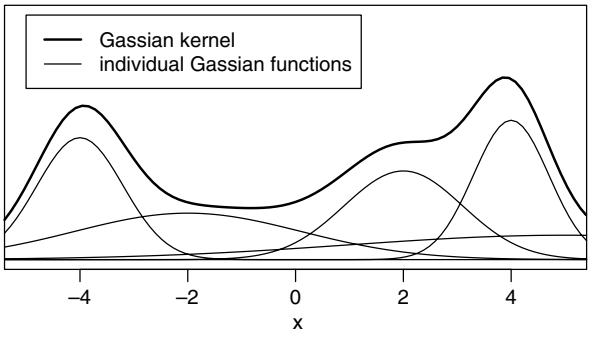

Fig. 2. Example of five Gaussian functions and their superposition–the resulting Gaussian kernel (illustration limited to the range $x \in [-5, 5]$).

[3] Such pseudo-random number generators are routinely available for most programming languages.

3.2. Pheromone representation in $\mathrm{ACO}_{\mathbb{R}}$

In ACO for combinatorial optimization, pheromone information is stored as a table. At each iteration, when choosing a component to be added to the current partial solution (according to Eq. (1)), an ant uses some of the values from that table as a discrete probability distribution (Fig. 1(a)). In the case of continuous optimization, the choice an ant makes is not restricted to a finite set (Fig. 1(b)). Hence, it is impossible to represent the pheromone in the form of a table. A different approach has to be adopted.

We use an idea similar to that proposed by Guntsch and Middendorf in *Population-Based ACO* (PB-ACO) (Guntsch and Middendorf, 2002). In PB-ACO the pheromone table is updated based on the components of good solutions found—just like in regular ACO. However, in regular ACO, the actual solutions found by the ants are discarded once the pheromone table has been updated. In contrast, PB-ACO keeps track of a certain number of the solutions used to update the pheromone table. Instead of using pheromone evaporation, the pheromone associated with the oldest solutions is eventually removed by performing a negative update on the pheromone table—thus canceling its influence.

In $\mathrm{ACO}_{\mathbb{R}}$ we also keep track of a number of solutions in a *solution archive* T. For each solution s_l to an n-dimensional problem, $\mathrm{ACO}_{\mathbb{R}}$ stores in T the values of its n variables and the value of the objective function $f(s_l)$. The ith variable of lth solution is hereby denoted by s_l^i. The structure of the solution archive T is presented in Fig. 3.

While in case of PB-ACO the components of the solutions are used directly to modify the pheromone table, in the continuous case we use them to dynamically generate probability density functions. In order to accomplish this, a method for generating a PDF based on a set of memorized solutions is defined. As indicated in Section 3.1 (Eq. (5)), the Gaussian kernel PDF G^i is parametrized by three vectors $\boldsymbol{\omega}$, $\boldsymbol{\mu}^i$, and $\boldsymbol{\sigma}^i$ (each of cardinality k). The solutions in the archive are used to calculate the values of these parameters, and hence shape the Gaussian kernel PDF used to guide the ants in their search process.

The number of solutions memorized in the archive is set to k and this parameter determines therefore the complexity of the PDF: There are k separate Gaussian functions making up the Gaussian kernel PDF.

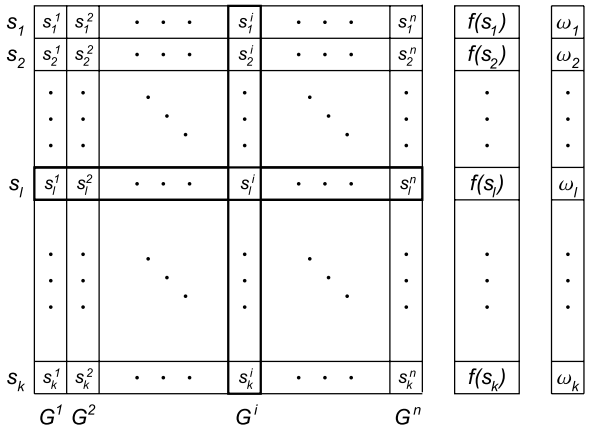

Fig. 3. The archive of solutions kept by $\mathrm{ACO}_{\mathbb{R}}$. The solutions are ordered in the archive according to their quality—i.e., for a minimization problem: $f(s_1) \leqslant f(s_2) \leqslant \cdots \leqslant f(s_l) \leqslant \cdots \leqslant f(s_k)$. Each solution has an associated weight ω proportion to the solution quality. Therefore, $\omega_1 \geqslant \omega_2 \geqslant \cdots \geqslant \omega_l \geqslant \cdots \geqslant \omega_k$. The PDF G^i is constructed using only the ith coordinates of all k solutions from the archive.

For each dimension $i = 1, \ldots, n$ of the problem, there is a different Gaussian kernel PDF G^i defined (see Fig. 3). For each such G^i, the values of the ith variable of all the solutions in the archive become the elements of the vector $\boldsymbol{\mu}^i$:

$$\boldsymbol{\mu}^i = \{\mu_1^i, \ldots, \mu_k^i\} = \{s_1^i, \ldots, s_k^i\}. \tag{6}$$

The vector of weights $\boldsymbol{\omega}$ is created in the following way. Each solution that is added to the archive T is evaluated and ranked (ties are broken randomly). The solutions in the archive are sorted according to their rank—i.e., solution s_l has rank l. The weight ω_l of the solution s_l is calculated according to the following formula:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} \mathrm{e}^{-\frac{(l-1)^2}{2q^2k^2}}, \tag{7}$$

which essentially defines the weight to be a value of the Gaussian function with argument l, mean 1.0, and standard deviation qk, where q is a parameter of the algorithm. When q is small, the best-ranked solutions are strongly preferred, and when it is large, the probability becomes more uniform. The influence of this parameter on $\mathrm{ACO}_{\mathbb{R}}$ is similar to adjusting the balance between the *iteration-best* and the *best-so-far* pheromone updates used in ACO. A more detailed analysis of the influence of the parameter q is presented in Section 5.3.

In order to find the final shape of each Gaussian kernel PDF G^i, the vector $\boldsymbol{\sigma}^i$ of the standard

deviations must still be defined. The detailed description of how this is accomplished is presented in the following section as part of the description of the solution construction process.

3.3. The $ACO_{\mathbb{R}}$ metaheuristic framework

In this section, we outline the $ACO_{\mathbb{R}}$ version of the three major algorithmic components of the ACO metaheuristic as presented in Algorithm 1.

AntBasedSolutionConstruction( ): Given decision variables X_i, $i = 1,\ldots,n$, an ant constructs a solution by performing n construction steps. At construction step i an ant chooses a value for variable X_i. As mentioned earlier, the Gaussian kernel PDF is composed of a number of regular Gaussian functions. The number of functions used is equal to the size k of the solution archive T. At construction step i, only the information about the ith dimension (i.e., decision variable X_i) is used. In this way, at each step i the resulting Gaussian kernel PDF G^i is a different one.

Following Eq. (5), in order to define the PDF G^i, the values of vectors $\boldsymbol{\mu}^i$, $\boldsymbol{\sigma}^i$, and $\boldsymbol{\omega}$ must be defined. While the creation of $\boldsymbol{\mu}^i$ and $\boldsymbol{\omega}$ has been discussed in Section 3.2, the computation of the standard deviation vector $\boldsymbol{\sigma}^i$ is the most complex issue. Before presenting how this is done in detail, we explain the practical implementation of Eq. (5).

In practice, the sampling process is accomplished as follows. First, the elements of the weight vector $\boldsymbol{\omega}$ are computed following Eq. (7). Then, the sampling is done in two phases. Phase one consists of choosing one of the Gaussian functions that compose the Gaussian kernel. The probability p_l of choosing the lth Gaussian function is given by:

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r}. \tag{8}$$

Phase two consists of sampling the chosen Gaussian function (i.e., at step i—function g_l^i). This may be done using a random number generator that is able to generate random numbers according to a parametrized normal distribution, or by using a uniform random generator in conjunction with, for instance, the Box–Muller method (Box and Muller, 1958). This two-phase sampling is equivalent to sampling the Gaussian kernel PDF G^i as defined in Eq. (5).

It is clear that at step i, the standard deviation needs only to be known for the single Gaussian function $g_l^i(x)$ chosen in phase one. Hence, we do not calculate the whole vector of standard deviations $\boldsymbol{\sigma}^i$, but only the σ_l^i that is needed.

The choice of the lth Gaussian function is done only once per ant, per iteration. This means that an ant uses the Gaussian functions associated with the single chosen solution s_l, that is, functions $g_l^i, i = 1,\ldots,n$, for constructing the whole solution in a given iteration. This allows exploiting the correlation between the variables, which is explained in detail in Appendix A. Of course, the actual Gaussian function sampled differs at each construction step, as for step i, $\mu_l^i = s_l^i$, and σ_l^i is calculated dynamically, as follows.

In order to establish the value of the standard deviation σ_l^i at construction step i, we calculate the average distance from the chosen solution s_l to other solutions in the archive, and we multiply it by the parameter ξ:

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{|s_e^i - s_l^i|}{k-1}. \tag{9}$$

The parameter $\xi > 0$, which is the same for all the dimensions, has an effect similar to that of the pheromone evaporation rate in ACO. The higher the value of ξ, the lower the convergence speed of the algorithm. While the rate of pheromone evaporation in ACO influences the long term memory—i.e., worse solutions are forgotten faster—ξ in $ACO_{\mathbb{R}}$ influences the way the long term memory is used—i.e., the search is less biased towards the points of the search space that have been already explored (and which are kept in the archive).

As we said, this whole process is repeated for each dimension $i = 1,\ldots,n$, and each time the average distance σ_l^i is calculated only with the use of the single dimension i. This ensures that the algorithm is able to adapt to linear transformations of the considered problem (e.g., moving from a sphere model to an ellipsoid, or rotating an ellipsoid).

PheromoneUpdate( ): As mentioned earlier, in case of $ACO_{\mathbb{R}}$, the pheromone information is stored as a *solution archive*. This implies that the pheromone update procedure has to perform some form of update on this archive.

The size k of the archive T is a parameter of the algorithm. However, k may not be smaller than the number of dimensions of the problem being solved.[4]

[4] This is due to the explicit handling of correlation among variables as explained in Appendix A. In order to be able to rotate the coordinate system properly, the number of points available has to be at least equal to the number of dimensions.

At the start of the algorithm, the solution archive T is initialized generating k solutions by uniform random sampling.

Pheromone update is accomplished by adding the set of newly generated solutions to the solution archive T and then removing the same number of worst solutions, so that the total size of the archive does not change. This process ensures that only the best solutions are kept in the archive, so that they effectively guide the ants in the search process.

DaemonActions( ): As part of this algorithmic block, the best solution found is updated, so that it may be returned once the termination condition is met. We do not apply any local search heuristics, though this could be easily done to improve the algorithm performance.

4. Positioning of $ACO_{\mathbb{R}}$

$ACO_{\mathbb{R}}$ is part of a rather large family of algorithms for continuous optimization. For this kind of problems, a number of methods have been proposed in the literature. They include some ant-related methods (Bilchev and Parmee, 1995; Monmarché et al., 2000; Dréo and Siarry, 2002), as well as a more generic *swarm* inspired method such as Particle Swarm Optimization (Kennedy and Eberhart, 1995). There are also many others: many metaheuristics have been originally developed for combinatorial optimization and later adapted to the continuous case. Examples include the Continuous Genetic Algorithm (CGA) (Chelouah and Siarry, 2000), Enhanced Simulated Annealing (ESA) (Siarry et al., 1997), or Enhanced Continuous Tabu Search (ECTS) (Chelouah and Siarry, 1999).

Additionally, there are also other methods that—similarly to ACO—explicitly use some notion of probability distribution estimation. Many of these algorithms have spawned from the general class of Evolutionary Algorithms (EAs). Examples include Evolutionary Strategies (ES) (Schwefel, 1981; Ostermeier et al., 1994; Hansen and Ostermeier, 2001), Iterated Density Estimation Algorithm (IDEA) (Bosman and Thierens, 2002), Mixed Bayesian Optimization Algorithm (MBOA) (Očenášek and Schwarz, 2002), or Population-Based Incremental Learning (PBIL) (Baluja and Caruana, 1995; Yuan and Gallagher, 2003). Some of them, similarly to ACO, have been initially used for combinatorial optimization, and only later adapted to handle continuous domains.

In addition to all the algorithms mentioned so far, there are also many gradient based algorithms for continuous optimization. They are fast, but they have some prerequisites. They are able to quickly find a local minimum, but they require the optimized function to be continuous and differentiable. Examples of such algorithms include the Newton method (Ralston and Rabinowitz, 1978), or the backpropagation algorithm (Rumelhart et al., 1986) routinely used for training artificial neural networks. The usefulness of gradient based algorithms is limited due to the prerequisites mentioned above. $ACO_{\mathbb{R}}$, as well as all other algorithms for continuous optimization mentioned earlier, do not have such limitations, which makes them much more general.

4.1. $ACO_{\mathbb{R}}$ and other swarm-based algorithms

The main type of swarm-based algorithms that we will refer to in this section are the ant-related algorithms. A single swarm-based algorithm that is not ant-related will be presented towards the end of this section.

As mentioned in Section 1, there have been previous attempts to apply ant-based optimization algorithms to the continuous domain. Some attempts were more successful than others, but none of them was an extension of ACO to the continuous domain. Rather, they were new algorithms that also drew their initial inspiration from the behavior of ants. In the following paragraphs, we shortly present these algorithms and indicate how they differ from $ACO_{\mathbb{R}}$.

One of the first attempts to apply an ant-related algorithm to the continuous optimization problems was Continuous ACO (CACO) (Bilchev and Parmee, 1995). In CACO the ants start from a point, called a *nest*, situated somewhere in the search space. The good solutions found are stored as a set of vectors, which originate in the nest. The ants at each iteration of the algorithm choose probabilistically one of the vectors. They then continue the search from the end-point of the chosen vector by making some random moves from there. The vectors are updated with the best results found. Although the authors of CACO claim that they draw inspiration from the original ACO formulation, there are important differences. They introduce the notion of *nest*, which does not exist in the ACO metaheuristic. Also, CACO does not perform an incremental construction of solutions, which is one

of the main characteristics of the ACO metaheuristic. CACO does not qualify therefore to be an extension of ACO.

Another ant-related approach to continuous optimization is the API algorithm (Monmarché et al., 2000). API does not claim to be based on the ACO metaheuristic. The ants perform their search independently, but starting from the same nest (the nest is moved periodically). The ants use only *tandem running*, a type of recruitment strategy. It is the only known algorithm among the ant-related algorithms published so far that allows to tackle both discrete and continuous optimization problems.

The third ant-based approach to continuous optimization is Continuous Interacting Ant Colony (CIAC) (Dréo and Siarry, 2002). CIAC uses two types of communication between ants: stigmergic information (spots of pheromone deposited in the search space) and direct communication between ants. The ants move through the search space being attracted by pheromone laid in spots, and guided by some direct communication between individuals. Although also CIAC claims to draw its original inspiration from ACO, the differences are many: there is direct communication between ants and no incremental construction of solutions. As CACO, also CIAC does not qualify as an extension of ACO.

Finally, as mentioned at the beginning of this section, there is a well-known *swarm-based* algorithm for continuous optimization that is not ant-related. It is called Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995). PSO works with a population of particles. These particles move in the search space with a certain velocity. The value and the direction of the velocity vector change according to the attractors in the search space. Each particle reacts to two such attractors. One is the best value found by the particle, and the other one is the best value found globally. PSO has been shown experimentally to perform well on many continuous optimization problems.

4.2. $ACO_{\mathbb{R}}$ and evolutionary algorithms

ACO for combinatorial optimization is similar to Evolutionary Algorithms (EAs) in many respects. Both ACO and EAs explicitly use some notion of probability distribution in order to find promising areas in the search space. This similarity is maintained when comparing $ACO_{\mathbb{R}}$ with EAs developed or adapted to continuous domains.

Kern et al. (2004) present an extensive comparison of several evolutionary algorithms dedicated to continuous optimization—from very simple ones to those that are quite advanced. We will now shortly present them.

The set of algorithms compared by Kern et al. contains three versions of Evolutionary Strategies (ES), and two other algorithms—the Mixed-Bayesian Optimization Algorithm (MBOA), and the Iterated Density Estimation Algorithm (IDEA). The simplest algorithm used in this comparison is $(1 + 1)$ ES (Kern et al., 2004). It is a simple ES with one parent generating one offspring per iteration. Only the individual representing the higher quality solution is kept. The next ES included in the comparison is Evolutionary Strategy with Cumulative Step Size Adaptation (CSA-ES) (Ostermeier et al., 1994; Kern et al., 2004). It adapts the global step size by using the path traversed by the parent population over a number of generations. The third ES considered is CMA-ES—ES with Covariance Matrix Adaptation (Hansen and Ostermeier, 2001; Kern et al., 2004). It is an extended version of CSA-ES, with de-randomized adaptation of the covariance matrix.

The first of the two algorithms that are not ES is IDEA, proposed by Bosman and Thierens (2002). It formalizes EDAs (Estimation of Distribution Algorithms) in continuous domains. To estimate the distribution of the parent population, IDEA exploits the fact that every multivariate joint probability distribution can be written as a *conditional factorization*: $P(x_i, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | x_{i+1}, x_{i+2}, \ldots, x_n)$. The probabilistic model of the parent population is rebuilt in every generation.

The last algorithm considered in this comparison is MBOA (Očenášek and Schwarz, 2002). It is a Bayesian network with local structures in the form of decision trees that captures the mutual dependencies among the parent individuals. The first EDA employing the Bayesian network model with decision trees was the hierarchical Bayesian Optimization Algorithm (hBOA) (Pelikan et al., 2000). MBOA is an extension of hBOA from binary to continuous domains. In fact, MBOA is able to deal with discrete and continuous variables simultaneously, just like $ACO_{\mathbb{R}}$.

Each of these algorithms is different, but they all use some way of learning and modeling explicitly probability distributions. There are two ways these algorithms use the probability distributions. All versions of the ES incrementally update their

probability distributions at each iteration. In contrast, IDEA and MBOA each time entirely rebuild them. $ACO_{\mathbb{R}}$ acts in this respect yet differently. Similarly to ACO for combinatorial optimization, in $ACO_{\mathbb{R}}$ ants use at each construction step a different, dynamically created probability distribution. This distribution depends on the previous construction steps and may be different for each ant. Hence, the $ACO_{\mathbb{R}}$ approach is closer to what IDEA and MBOA do, but it is even more fine-grained—several dynamically created probability distributions are used during one iteration.

All of the probability-learning algorithms compared by Kern et al. use some form of Gaussian function for modeling the probability distributions. $(1 + 1)$ES and CSA-ES use isotropic Gaussian distributions. IDEA uses one (or more—a mixture, if clustering is enabled) arbitrary Gaussian distribution. MBOA uses a concept somewhat similar to the one used by $ACO_{\mathbb{R}}$—Gaussian kernel distribution, but defined on partitions of the search space. $ACO_{\mathbb{R}}$ in turn uses a set of single-dimension Gaussian kernel distributions. Each such distribution is used for a different construction step during one iteration. The main characteristic of $ACO_{\mathbb{R}}$ is that it uses a different distribution for each of the dimensions. Each such Gaussian kernel PDF consists of several superimposed single-dimension Gaussian functions.

5. Experimental setup and results

In this section, we present the experimental setup for evaluating the performance of $ACO_{\mathbb{R}}$ and the results obtained. In order to have an overview of the performance of $ACO_{\mathbb{R}}$ in comparison to other methods for continuous optimization, we use the typical benchmark test functions that have been used in the literature for presenting the performance of different methods and algorithms.

Obviously, it is impractical to compare $ACO_{\mathbb{R}}$ to every method that has been used for continuous optimization in the past. Hence, we decided to limit our comparison to other metaheuristics used for this purpose. We then decided to divide those metaheuristics into three groups, based on their similarity to $ACO_{\mathbb{R}}$:

- Probability-learning methods—methods which explicitly model and sample probability distributions.
- Ant-related methods—methods that claim to draw inspiration from the behavior of ants.
- Other metaheuristics originally developed for combinatorial optimization and later adapted to continuous domains.

We have used a slightly different experimental methodology for each comparison in order to make the results obtained by $ACO_{\mathbb{R}}$ as comparable as possible to those obtained with the other methods.

It is important to emphasize that—unlike combinatorial optimization—the comparison of algorithms for continuous optimization is usually *not* done based on CPU time. In case of combinatorial optimization, usually each algorithm is given the same amount of CPU time and the results obtained within that time are compared. This makes the comparison of different algorithms complicated, as the CPU time depends significantly on the programming language used, the compiler, the skills of the programmer, and finally also on the machine(s) used for running the experiments. Hence, in case of combinatorial optimization it is strongly recommended to re-implement all the algorithms used in the comparison in order to make it fair. This still does not guarantee an entirely fair comparison, as it is difficult to ensure that the same amount of effort is put into optimization of the code of all the implemented algorithms.[5]

In contrast, the great majority of the papers on continuous optimization algorithms use as criterion of comparison the *number of function evaluations* needed to achieve a certain solution quality (Kern et al., 2004; Bilchev and Parmee, 1995; Monmarché et al., 2000; Dréo and Siarry, 2004). Such an approach gives several key advantages: it solves the problem of the algorithms being implemented using different programming languages; it is insensitive to the code-optimization skills of the programmer (or to the compiler used); and it allows comparing easily the results obtained on different machines. The drawback of this approach is that it does not take into consideration the time-complexity of the algorithms compared. However, in view of the other numerous disadvantages of using the CPU time as a criterion, it is an acceptable methodology, and we adopt it in this paper.

The use of the number of function evaluations as a criterion allows us to run the experiments only with $ACO_{\mathbb{R}}$ and compare the results obtained to

[5] Automatic parameter tuning procedures such as F-Race (Birattari et al., 2002; Birattari, 2005) can help in this respect.

Table 1
Summary of the test functions used for comparing $ACO_{\mathbb{R}}$ with other probability learning methods

Function	Formula
Plane (PL) $\vec{x} \in [0.5, 1.5]^n,\ n = 10$	$f_{\mathrm{PL}}(\vec{x}) = x_1$
Diagonal plane (DP) $\vec{x} \in [0.5, 1.5]^n,\ n = 10$	$f_{\mathrm{DP}}(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$
Sphere (SP) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{\mathrm{SP}}(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Ellipsoid (EL) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{\mathrm{EL}}(\vec{x}) = \sum_{i=1}^{n} \left(100^{\frac{i-1}{n-1}} x_i\right)^2$
Cigar (CG) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{\mathrm{CG}}(\vec{x}) = x_1^2 + 10^4 \sum_{i=2}^{n} x_i^2$
Tablet (TB) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{\mathrm{TB}}(\vec{x}) = 10^4 x_1^2 + \sum_{i=2}^{n} x_i^2$
Rosenbrock (Rn) $\vec{x} \in [-5, 5]^n, n = 10$	$f_{\mathrm{Rn}}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2$

We used $n = 10$ dimensions for all the test functions listed here.

those found in the literature. Additionally, in order to ensure a fair comparison, we replicate carefully the experimental setup (in particular: initialization interval, parameter tuning methodology, and termination condition) used by the competing algorithms.

5.1. $ACO_{\mathbb{R}}$ compared to the probability-learning methods

There are many metaheuristics that explicitly model and sample probability distributions. In this comparison we use the algorithms tested by Kern et al. (2004), which have been already described in Section 4.2.

The test functions used for comparison range from very simple to quite complex, and were chosen by Kern et al. for their particular characteristics. In particular, they test the algorithms' robustness when applied to linear transformations of the considered problem. For a fair comparison of $ACO_{\mathbb{R}}$'s performance, we have used the same test functions. They are listed in Table 1. *Plane* and *Diagonal Plane* functions are maximization problems (the goal is to reach $\epsilon_{\max} = 10^{10}$), the remaining are minimization problems (required accuracy is $\epsilon_{\min} = 10^{-10}$). In addition to the functions listed in Table 1, the comparison also uses randomly rotated versions of the *Ellipsoid*, *Cigar*, and *Tablet* functions. In all the experiments we used 10-dimensional versions of the functions. The performance was judged based on the number of function evaluations needed to reach the stopping condition. The stopping condition used was the required accuracy: $f > \epsilon_{\max}$ for maximization problems, and $|f - f^*| < \epsilon_{\min}$ for minimization problems, where f is the value of the best solution found by $ACO_{\mathbb{R}}$, and f^* is the (known *a priori*) optimal solution for the given test problem.

When tackling continuous optimization *test* problems (i.e., those for which the optimal solution is known *a priori*), one has to do the initialization with caution.[6] In fact, it has been shown in the literature that if the initialization is done close to the optimum, or it is *symmetric* around the optimum, it may introduce an undesired bias (Fogel and Bayer, 1995). It has also been demonstrated that the results obtained by some algorithms differ significantly, when the symmetric and asymmetric initialization are used (Eiben and Bäck, 1997; Chellapilla and Fogel, 1999).

Due to these issues, special care must be taken in order to use (when possible) initializations that do not introduce bias. Such an approach is often called *skewed initialization* (Eiben and Bäck, 1997; Deb et al., 2002). Also, any comparison of the continuous optimization algorithms should explicitly take into account the initialization used by each of the algorithms. In all our test runs we use initialization intervals identical to those used by the methods we compare to.

In order to compare the performance of $ACO_{\mathbb{R}}$ to that of the algorithms presented in Kern et al. (2004), we have adopted the same methodology for conducting the experiments. We have done 20 independent runs on each of the test problems. Concerning parameters tuning, Kern et al. used the parameters as suggested by the authors, with the exception of the size of the population[7] used. The latter was chosen separately for each

[6] In the case of real-world problems, the optimal solutions are often not known, and hence the initialization intervals have to be chosen based on some other estimates or even at random.

[7] In case of the evolutionary algorithms, there is the notion of *population* size. In case of $ACO_{\mathbb{R}}$, the corresponding parameter is the *archive* size.

Table 2
Summary of the parameters used by $ACO_{\mathbb{R}}$

Parameter	Symbol	Value
No. of ants used in an iteration	m	2
Speed of convergence	ξ	0.85
Locality of the search process	q	10^{-4}
Archive size	k	50

The solution archive size varied depending on the test function, as done by Kern et al. (2004).

algorithm-problem pair—the smallest population from the set $p \in [10, 20, 50, 100, 200, 400, 800, 1600, 3200]$ was chosen, which still allowed to achieve the required accuracy in all the runs. The summary of the parameters we used for $ACO_{\mathbb{R}}$ is presented in Table 2. The archive size $k = 50$ was used for all the test problems.

Table 3 presents the results obtained by $ACO_{\mathbb{R}}$ compared to those obtained by the algorithms tested by Kern et al. For each test problem, the relative median performance for all the algorithms is reported, 1.0 being the best algorithm (lowest median number of function evaluations). Numbers for the other algorithms indicate *how many times* larger was their median number of function evaluations in relation to the best one on a given test function. For the best algorithm also the actual median number of function evaluations is supplied in parentheses.

The performance of $ACO_{\mathbb{R}}$ is quite good. It achieves the best result in four out of 10 test problems. Also, when it is not the best, it is only slightly worse than the best algorithm (CMA-ES). Unlike some of the competing algorithms, $ACO_{\mathbb{R}}$ is performing well in case of both maximization (Plane and Diagonal Plane) and minimization problems. It is able to adjust well to problems that are scaled differently in different directions (such as Ellipsoid, Cigar, and Tablet functions), and the rotation of the test function does not have any impact on the performance. In this respect only CMA-ES performs similarly well.

Due to unavailability of full result sets and missing results of some algorithms for some test functions, it is impossible to do any serious statistical significance analysis. In order to enable future researchers to perform more statistically sound comparisons, the full set of results obtained with $ACO_{\mathbb{R}}$ is available online.[8]

[8] http://iridia.ulb.ac.be/~ksocha/aco_r.html.

5.2. $ACO_{\mathbb{R}}$ compared to other ant-related approaches and other metaheuristics

As we have mentioned earlier, there were other ant-related methods proposed for continuous optimization in the past. The very first one—called Continuous ACO (CACO)—was proposed by Bilchev and Parmee (1995), and also used later (Wodrich and Bilchev, 1997; Mathur et al., 2000). Others include the API algorithm by Monmarché et al. (2000), and Continuous Interacting Ant Colony (CIAC), proposed by Dréo and Siarry (2002, 2004). These algorithms have been already described in Section 4.1. They were tested by their authors on some classical test functions and compared with other metaheuristics that had been primarily developed for combinatorial optimization and later adapted to the continuous domain. The continuous versions of these metaheuristics include in particular Continuous Genetic Algorithm (CGA) (Chelouah and Siarry, 2000), Enhanced Continuous Tabu Search (ECTS) (Chelouah and Siarry, 1999), Enhanced Simulated Annealing (ESA) (Siarry et al., 1997), and Differential Evolution (DE) (Storn and Price, 1995).

In these comparisons, the parameters chosen by the authors of the competing algorithms were essentially picked by a simple trial and error procedure. Hence, we have also refrained from doing an extensive parameter tuning. Instead, we used almost identical parameters to those used earlier for comparing $ACO_{\mathbb{R}}$ with probability-learning methods. The only exception was parameter q, which required a slightly larger value in order to increase the robustness of $ACO_{\mathbb{R}}$ on the multi-modal functions used in this set of experiments. The value $q = 0.1$ was used. More analysis of the influence of parameter q on the performance of $ACO_{\mathbb{R}}$ is provided in Section 5.3.

To compare $ACO_{\mathbb{R}}$ with all these algorithms, we have run $ACO_{\mathbb{R}}$ on a number of test functions used by the other algorithms, and we followed the original experimental setup in terms of the initialization interval and required accuracy. The list of test functions on which we run $ACO_{\mathbb{R}}$, along with the number of dimensions used and the initialization interval, is presented in Tables 4 and 5. A more detailed description of the test functions used may be found in the literature (Björkman and Holmström, 1999; Chelouah and Siarry, 2000; Dréo and Siarry, 2004). Following the experimental setup described in the literature, we performed 100 inde-

Table 3
Results obtained by $ACO_{\mathbb{R}}$ compared to those obtained by other probability-learning algorithms—based on Kern et al., 2004

Test Function	$ACO_{\mathbb{R}}$	(1 + 1)ES	CSA-ES	CMA-ES	IDEA	MBOA
Plane	**1.0** (175)	4.5	7.2	6.3	∞	4970
Diagonal plane	**1.0** (170)	4.9	7.4	6.4	∞	241
Sphere	1.1	**1.0** (1370)	1.6	1.3	5.0	48
Ellipsoid	2.6	66	110	**1.0** (4450)	1.6	14
Cigar	1.4	610	800	**1.0** (3840)	4.6	12
Tablet	**1.0** (2567)	46	65	1.7	2.9	24
Rot. Ellipsoid	2.8	64	110	**1.0** (4490)	13	*1800
Rot. Cigar	1.4	600	800	**1.0** (3840)	38	*2100
Rot. Tablet	**1.0** (2508)	44	63	1.7	12	*1600
Rosenbrock	*1.1	*51	180	**1.0** (7190)	*210	*1100

Reported is the relative median number of function evaluations. The actual median number of function evaluations is given in parentheses only for the best performing algorithm on a given problem. Results marked with * indicate that the required accuracy was not reached in every run.

Table 4
First part of the test functions used for comparing $ACO_{\mathbb{R}}$ to other ant-related algorithms and other metaheuristics adapted for continuous optimization

Function	Formula
Branin RCOS (RC) $\vec{x} \in [-5,15]^n,\ n=2$	$f_{\mathrm{RC}}(\vec{x}) = \left(x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_i + 10$
$B_2\ \vec{x} \in [-100,100]^n,\ n=2$	$f_{B_2}(\vec{x}) = x_1^2 + 2x_2^2 - \frac{3}{10}\cos(3\pi x_1) - \frac{2}{5}\cos(4\pi x_2) + \frac{7}{10}$
Easom (ES) $\vec{x} \in [-100,100]^n,\ n=2$	$f_{\mathrm{ES}}(\vec{x}) = -\cos(x_1)\cos(x_2)e^{-((x_1-\pi)^2+(x_2-\pi)^2)}$
Goldstein and Price (GP) $\vec{x} \in [-2,2]^n,\ n=2$	$f_{\mathrm{GP}}(\vec{x}) = (1+(x_1+x_2+1)^2(19-14x_1+13x_1^2-14x_2+6x_1x_2+3x_2^2))\cdot (30+(2x_1-3x_2)^2(18-32x_1+12x_1^2-48x_2-36x_1x_2+27x_2^2))$
Martin and Gaddy (MG) $\vec{x} \in [-20,20]^n,\ n=2$	$f_{\mathrm{MG}}(\vec{x}) = (x_1-x_2)^2 + \left(\frac{x_1+x_2-10}{3}\right)^2$
Rosenbrock (R_n) $\vec{x} \in [-5,10]^n,\ n=2,5$	$f_{R_n}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i-1)^2$
Zakharov (Z_n) $\vec{x} \in [-5,10]^n,\ n=2,5$	$f_{Z_n}(\vec{x}) = \left(\sum_{j=1}^n x_j^2\right) + \left(\sum_{j=1}^n \frac{jx_j}{2}\right)^2 + \left(\sum_{j=1}^n \frac{jx_j}{2}\right)^4$
De Jong (DJ) $\vec{x} \in [-5.12,5.12]^n,\ n=3$	$f_{\mathrm{DJ}}(\vec{x}) = x_1^2 + x_2^2 + x_3^2$
Griewangk (GR_n) $\vec{x} \in [-5.12,5.12]^n,\ n=10$	$f_{\mathrm{GR}_n}(\vec{x}) = \left(\frac{1}{10} + \left(\sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)\right)^{-1}$
Sphere model (SM) $\vec{x} \in [-5.12,5.12]^n,\ n=6$	$f_{\mathrm{SM}}(\vec{x}) = \sum_{i=1}^n x_i^2$

pendent runs, and we used the following stopping criterion (as used by the other algorithms in this comparison):

$$|f - f^*| < \epsilon_1 f + \epsilon_2, \tag{10}$$

where f is the value of the best solution found by $ACO_{\mathbb{R}}$, f^* is the (known *a priori*) optimal value for the given test problem, and ϵ_1 and ϵ_2 are respectively the relative and absolute errors. For all the test runs of $ACO_{\mathbb{R}}$, we used $\epsilon_1 = \epsilon_2 = 10^{-4}$, following the values reported in the literature.

Tables 6 and 7 present the results obtained by $ACO_{\mathbb{R}}$, and respectively other ant-related algorithms and other metaheuristics. Some of the test functions used in this section are multi-modal—they have many local optima, where algorithms may get stuck. Hence, the results presented not only give an overview of the mean performance (i.e., number of function evaluations), but also give the success rate—percentage of successful runs, when the algorithm found the global optimum.

When compared with other ant-related methods (Table 6), $ACO_{\mathbb{R}}$ is a clear winner—all other algorithms in this category require many more function evaluations in order to reach the required accuracy. $ACO_{\mathbb{R}}$ is simply a much more effective approach. As mentioned earlier, serious statistical analysis of the results is not possible due to unavailability of the full data result sets for the other algorithms.

When $ACO_{\mathbb{R}}$ is compared to other metaheuristics adapted for continuous domains, such as CGA, ECTS, or ESA (Table 7), it is not so clear anymore which of the algorithms is best. Almost each of the algorithms presented is best-performing for at least some of the test problems (with the exception of ESA). While $ACO_{\mathbb{R}}$ is the winner for Easom, both

Table 5
Second part of the test functions used for comparing $ACO_{\mathbb{R}}$ to other ant-related algorithms and other metaheuristics adapted for continuous optimization

Function	Formula
Hartmann ($H_{3,4}$) $\vec{x} \in [0,1]^n,\ n = 4$	$f_{H_{3,4}}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \begin{Bmatrix} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{Bmatrix},\ c_i = \begin{Bmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{Bmatrix},$ $p_{ij} = \begin{Bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0381 & 0.5743 & 0.8828 \end{Bmatrix}$
Hartmann ($H_{6,4}$) $\vec{x} \in [0,1]^n,\ n = 6$	$f_{H_{6,4}}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \begin{Bmatrix} 10.00 & 3.00 & 17.0 & 3.50 & 1.50 & 8.00 \\ 0.05 & 10.00 & 17.0 & 0.10 & 8.00 & 14.00 \\ 3.00 & 3.50 & 1.70 & 10.00 & 17.00 & 8.00 \\ 17.00 & 8.00 & 0.05 & 10.00 & 0.10 & 14.00 \end{Bmatrix},\ c_i = \begin{Bmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{Bmatrix},$ $p_{ij} = \begin{Bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{Bmatrix}$
Shekel ($S_{4,k}$,$k = 5,7,10$) $\vec{x} \in [0,10]^n,\ n = 4$	$f_{S_{4,k}}(\vec{x}) = -\sum_{i=1}^{k} ((\vec{x} - \vec{a}_i)^{\mathrm{T}}(\vec{x} - \vec{a}_i) + c_i)^{-1}$ $a_{ij} = \begin{Bmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 5.0 & 3.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{Bmatrix},\ c_i = \begin{Bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{Bmatrix}$

Table 6
Results obtained by $ACO_{\mathbb{R}}$ compared to those obtained by other ant-related algorithms

Test function	$ACO_{\mathbb{R}}$	CACO	API	CIAC
Rosenbrock (R_2)	**1.0** (820)	8.3	12	14
Sphere	**1.0** (781)	28	13	64
Griewangk (GR_{10})	**1.0** [61%] (1390)	36	–	36 [52%]
Goldstein and Price	**1.0** (384)	14	–	61 [56%]
Martin and Gaddy	**1.0** (345)	5	–	34 [20%]
B_2	**1.0**(544)	–	–	22
Rosenbrock (R_5)	**1.0** [97%] (2487)	–	–	16 [90%]
Shekel ($S_{4,5}$)	**1.0** [57%] (787)	–	–	50 [5%]

Reported is the relative mean number of function evaluations. The actual mean number of function evaluations is given in parentheses only for the best performing algorithm on a given problem. Numbers in square brackets indicate the percentage of successful runs (i.e., when the algorithm did not get stuck in a local optimum). When the percentage is not given—all the runs were successful. Note that for some algorithms, the results on some test functions were not available.

Hartmann, Griewangk, and one Zakharov problem, CGA is the leader for B_2 and all three Shekel problems. In turn ECTS is the best one on Branin RCOS, Goldstein and Price, one Zakharov, and both Rosenbrock problems. ESA is the only one that is not best on any of the problems, and results

Table 7
Results obtained by $ACO_{\mathbb{R}}$ compared to those obtained by other metaheuristics adapted to continuous domains

Test function	$ACO_{\mathbb{R}}$	CGA	ECTS	ESA	DE
Branin RCOS	3.5	2.5	**1.0** (245)	–	–
B_2	1.3	**1.0** (430)	–	–	–
Easom	**1.0** [98%] (772)	1.9	–	–	–
Goldstein and Price	1.7	1.8	**1.0** (231)	3.4	–
Rosenbrock (R_2)	1.7	2.0	**1.0** (480)	1.7	1.3
Zakharov (Z_2)	1.5	3.2	**1.0** (195)	81	–
De Jong	1.0	1.9	–	–	**1.0** (392)
Hartmann ($H_{3,4}$)	**1.0** (342)	1.7	1.6	2.0	–
Shekel ($S_{4,5}$)	1.3 [57%]	**1.0** [76%] (610)	1.4 [75%]	1.9 [54%]	–
Shekel ($S_{4,7}$)	1.1 [79%]	**1.0** [83%] (680)	1.3 [80%]	1.8 [54%]	–
Shekel ($S_{4,10}$)	1.1 [81%]	**1.0** [83%] (650)	1.4 [80%]	1.8 [50%]	–
Rosenbrock (R_5)	1.2 [97%]	1.9	**1.0** (2142)	2.5	–
Zakharov (Z_5)	**1.0** (727)	1.9	3.1	96	–
Hartmann ($H_{6,4}$)	**1.0** (722)	1.3	2.1	3.7	–
Griewangk (Gr_{10})	**1.0** [61%] (1390)	–	–	–	9.2

Reported is the relative mean number of function evaluations. The actual mean number of function evaluations is given in parentheses only for the best performing algorithm on a given problem. Numbers in square brackets indicate the percentage of successful runs. When the percentage is not given—all the runs were successful. Note that for some algorithms, the results on some test functions were not available.

of DE—although promising—are available for only very few problems.

The differences in performance between the metaheuristics are however rather small—they rarely exceed the factor of 2.0. It may be hence concluded that while each of these algorithms is different, they all perform similarly well, including $ACO_{\mathbb{R}}$ proposed in this paper. For particular real-world applications some of them may be better suited than others. However, the differences are not large, and it is not trivial to say when any of these algorithms should be preferred over the others. Again, as mentioned earlier, serious statistical analysis of the results is not possible due to unavailability of the full result data sets for the other algorithms.

5.3. Diversification versus intensification

When tackling multi-modal test functions, it is important that the algorithm is able to avoid being stuck in one of the local optima. An algorithm must use some strategy to *diversify* the search in such a way that it does not get stuck in a local optimum, and yet it is able to converge once the global optimum has been found.

These in fact are two contradictory goals. On one hand, an algorithm is expected to converge as fast as possible, while on the other hand, it is expected *not to converge* entirely to a local optimum. The fundamental problem is that an algorithm *does not know* if a given promising region contains a local or a global optimum. Hence, an algorithm has to make *an intelligent guess*, whether to focus on diversification (higher robustness), or intensification (higher convergence speed—higher efficiency).

Algorithms proposed for continuous optimization deal with this problem in various ways. Some just ignore it (like the simple $(1+1)$ES), but this usually does not give good results. Others explicitly divide the operation of the algorithm into the *diversification* and *intensification* phases—e.g., CGA, ECTS, or CIAC. Finally, some algorithms use one or more parameters in order to define the balance between diversification and intensification. Such an approach is used for instance by CSA-ES, CMA-ES, IDEA, and $ACO_{\mathbb{R}}$.

Usually, parameters such as learning rate and population size are those that most influence the robustness of the algorithm. In the case of $ACO_{\mathbb{R}}$, they also play some role—i.e., the slower the learning rate and the larger the solution archive size, the more robust is the algorithm, but the slower is the convergence speed. In $ACO_{\mathbb{R}}$, there is also another parameter specifically designed to control the diversification of the search process—parameter q.

When q approaches *0*, it means that only the Gaussian function associated with *the best solution found so far* is used for generating further solutions by the ants. Following Eqs. (7) and (8), for a given parameter q and size of the solution archive k, the

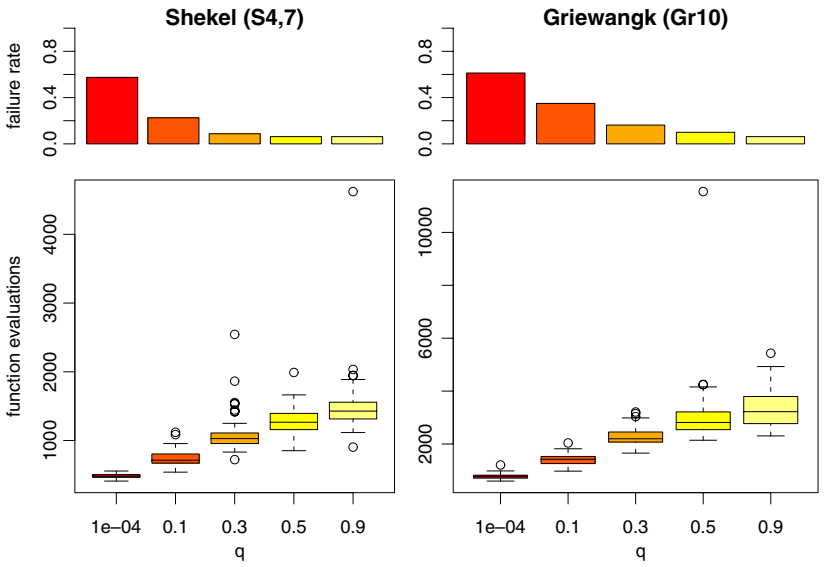

Fig. 4. Relationship between the robustness of the $ACO_{\mathbb{R}}$ algorithm and its efficiency. For each of the two test functions, and for each value of the parameter q tested, the failure rate (upper part) and the distribution of the number of function evaluations needed to reach the required accuracy (lower part) are given. The size of the solution archive was fixed at $k = 100$, and $m = 2$ ants were used in all runs.

probability p_{qk} of choosing one of the $q \cdot k$ highest ranking solutions as the base for the PDF is: $p_{qk} \approx 0.68$ (and respectively $p_{2qk} \approx 0.95$). This is due to the characteristic of the normal distribution: around 68% of the samples fall inside the interval $(-\sigma, \sigma)$ around the mean and respectively 95% in the interval $(-2\sigma, 2\sigma)$. For instance, for $q = 0.1$ and $k = 50$ (as used in experiments in Section 5.2), one of the 5 highest ranking solutions will be used with probability 0.68, and one of the 10 highest ranking solutions with probability 0.95.

When using larger q, the algorithm samples the search space based on a larger number of reasonably good solutions, rather than only on the best one found so far. The search is hence more diversified and the algorithm performs more robustly. Unfortunately, as already said, higher robustness usually means lower efficiency—slower convergence speed. This is illustrated in Fig. 4, which shows the failure rate (upper part) and the distribution of number of function evaluations for different values of the parameter q (lower part), for two typical multi-modal test functions—Shekel ($S_{4,7}$) and Griewangk (GR_{10}). The distribution of the number of function evaluations is given only for the successful runs. It is presented in the form of box-plots—the box is drawn between the first and the third quartile of the distribution, with median and outliers indicated.

6. Conclusions

We have presented in this paper a straightforward way of extending Ant Colony Optimization to continuous domains. We have discussed the idea and shown its implementation. $ACO_{\mathbb{R}}$ is a direct extension of ACO, and it is the first ant-based algorithm for continuous optimization which fits in the ACO framework.

We have discussed how $ACO_{\mathbb{R}}$ is situated within the (rather large) family of heuristic algorithms for continuous optimization. We have tested the performance of $ACO_{\mathbb{R}}$ against a substantial number of other algorithms and approaches. The results obtained show that $ACO_{\mathbb{R}}$ may be considered a competitive approach. Additionally, the performance of $ACO_{\mathbb{R}}$ may be adapted according to the needs to either show more robustness or higher efficiency.

$ACO_{\mathbb{R}}$, when compared to other probability-learning methods, proved to be the best on four out of 10 test problems. On the others, the quality of the solutions found was not significantly worse than the state-of-the-art. Also, $ACO_{\mathbb{R}}$ is a clear winner when compared to other ant-related algorithms for continuous optimization that were proposed in the past. When compared to these methods, $ACO_{\mathbb{R}}$ was better by almost two orders of magnitude. Finally, when compared to other metaheuris-

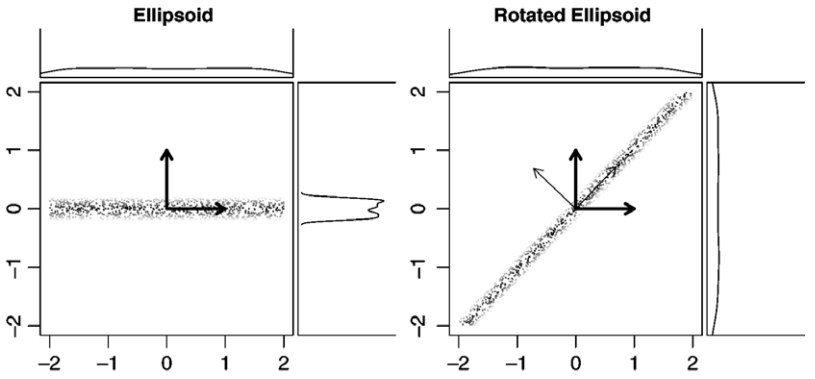

Fig. A.1. Example of the Ellipsoid function not rotated (left) and rotated by 45° (right). It is illustrated by 10 000 points—of which only the best 1000 are visible (the darker the point the higher the rank). The original coordinate system has been marked in bold, and the optimal one is also indicated on the right plot. Also, the examples of the Gaussian kernel PDFs as generated using the default coordinate system, are given on the right and above.

tics adapted to continuous optimization, $ACO_{\mathbb{R}}$ was the winner in one-third of the test problems and performed not much worse on the others.

The way ACO is extended to $ACO_{\mathbb{R}}$ allows it not only to handle the pure continuous problems, but also makes it possible to tackle mixed-variable optimization problems. Such ability can be matched by only few other approaches. Research into the performance of $ACO_{\mathbb{R}}$ as a mixed-variable optimizer is ongoing, and will be the subject of future publications.

Acknowledgements

Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Research Director. This work was supported by the "ANTS" project, an "Action de Recherche Concertée" funded by the Scientific Research Directorate of the French Community of Belgium.

Appendix A. Variable correlation handling

ACO algorithms in general do not exploit correlation information between different decision variables (or components). In $ACO_{\mathbb{R}}$, due to the specific way the pheromone is represented (i.e., as the solution archive), it is in fact possible to take into account the correlation between the decision variables. Consider Fig. A.1, where the same Ellipsoid test function[9] is presented—not rotated (left), and then randomly rotated (right). The test function is presented as the $ACO_{\mathbb{R}}$ algorithm sees it—as a set of points representing different solutions found by the ants and stored in the solution archive. The darker the point, the higher the quality of the solution (and the higher its rank). While on the left plot the variables are not correlated (i.e., for good solutions, the value of one coordinate does not depend on the value of the other coordinate), on the right plot they are highly correlated.

The default coordinate system that corresponds to the set of the original decision variables x_i, $i = 1, 2$, is marked in bold. It is clear that the axes of that coordinate system align well with the scaling of the test function in the left plot. The example Gaussian kernel PDFs for both of the dimensions are indicated on the right and above the plot. Clearly a new solution generated using them would fall somewhere in the promising region. In contrast, in the case of the rotated Ellipsoid function presented on the right plot, the PDFs created with the default coordinate system cover roughly the whole search space (here $\mathbf{D} = [-2, 2]^2$). If sampling was done based on the original coordinate system, the points sampled would most likely be far from being good. The optimal coordinate system that should be used by the ants in this case is also indicated on the right plot. If ants used that coordinate system instead of the original one, the sampling would be as efficient as in the case of the non-rotated Ellipsoid function.

Our $ACO_{\mathbb{R}}$ algorithm dynamically adapts the coordinate system used by each ant in order to minimize the correlation between different decision

[9] See Table 1 for the definition of the Ellipsoid function.

variables. The adaptation of the coordinate system is actually accomplished by expressing the set of decision variables $\mathbf{X}$ with temporary variables Z_i, $i = 1, \ldots, n$ that are linear combinations of X_i, $i = 1, \ldots, n$. The following paragraphs shortly present how this is done.

An obvious choice for adapting the coordinate system to the distribution of the solutions in the archive is the well-known technique of principal components analysis (PCA). For details we refer an interested reader to (Hastie et al., 2001). PCA performs a statistical analysis of the solutions in the archive in order to distinguish the principal components. However, due to the fact that PCA is deterministic, for most non-trivial problems PCA is not robust enough and often leads to stagnation.

The mechanism that we designed instead is relatively simple. Each ant at each step of the construction process chooses a *direction* in the search space. The direction is chosen by randomly selecting a solution s_u that is reasonably far away from the solution s_l chosen earlier as the mean of the PDF. Then, the vector $\vec{s_l s_u}$ becomes the chosen direction. The probability of choosing solution s_u at step i (having chosen earlier solution s_l as the mean of the PDF) is the following:

$$p_i(s_u|s_l) = \frac{d_i(s_u, s_l)^4}{\sum_{r=1}^{k} d_i(s_r, s_l)^4}, \qquad \text{(A.1)}$$

where the function $d_i(\cdot,\cdot)$ returns the Euclidean distance in the $(n - i + 1)$-dimensional search subspace[10] between two members of the solution archive T. Once this vector is chosen, the new orthogonal basis for the ant's coordinate system is created using the Gram–Schmidt process (Golub and van Loan, 1989). It takes as input all the (already orthogonal) directions chosen in earlier ant's steps and the newly chosen vector. The remaining missing vectors (for the remaining dimensions) are chosen randomly. Then, all the current coordinates of all the solutions in the archive are rotated and recalculated according to this new orthogonal base resulting in the set of new temporary variables Z_i, $i = 1, \ldots, n$.

At the end of the solution construction process, the chosen values of the temporary variables Z_i, $i = 1, \ldots, n$ are converted back into the original coordinate system, giving rise to a set of values for the original decision variables X_i, $i = 1, \ldots, n$.

[10] At step i, only dimensions i through n are used.

References

Baluja, S., Caruana, R., 1995. Removing the genetics from the standard genetic algorithm. In: Prieditis, A., Russel, S. (Eds.), Twelth International Conference on Machine Learning. Morgan Kaufmann Publishers, San Francisco, CA, pp. 38–46.

Bilchev, G., Parmee, I.C., 1995. The ant colony metaphor for searching continuous design spaces. In: Fogarty, T.C. (Ed.), Proceedings of the AISB Workshop on Evolutionary Computation, vol. 993 of LNCS. Springer-Verlag, Berlin, Germany, pp. 25–39.

Birattari, M., 2005. The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective. Ph.D. thesis, vol. 292 of Dissertationen zur Künstlichen Intelligenz. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany.

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., 2002. A Racing Algorithm for Configuring Metaheuristics. In: Langdon, W.B. et al. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufman, San Francisco, CA, pp. 11–18.

Björkman, M., Holmström, K., 1999. Global optimization using the DIRECT algorithm in Matlab. Advanced Modeling and Optimization 1 (2), 17–37.

Blum, C., 2004. Theoretical and Practical Aspects of Ant Colony Optimization. Ph.D. thesis, vol. 282 of Dissertationen zur Künstlichen Intelligenz. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany.

Blum, C., Sampels, M., 2004. An ant colony optimization algorithm for shop scheduling problems. Journal of Mathematical Modelling and Algorithms 3 (3), 285–308.

Bosman, P.A.N., Thierens, D., 2002. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In: Pelikan, M., Mühlenbein, H., Rodriguez, A.O. (Eds.), Proceedings of OBUPM Workshop at GECCO-2000. Morgan-Kaufmann Publishers, San Francisco, CA, pp. 197–200.

Box, G.E.P., Muller, M.E., 1958. A note on the generation of random normal deviates. Annals of Mathematical Statistics 29 (2), 610–611.

Chellapilla, K., Fogel, D.B., 1999. Fitness distribution in evolutionary computation: Analysis of local extrema in the continuous domain. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC-1999). IEEE Press, Piscataway, NJ, pp. 1885–1892.

Chelouah, R., Siarry, P., 1999. Enhanced continuous tabu search. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (Eds.), Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Boston, MA, pp. 49–61 (Chapter 4).

Chelouah, R., Siarry, P., 2000. A continuous genetic algorithm designed for the global optimization of multimodal functions. Journal of Heuristics 6, 191–213.

Costa, D., Hertz, A., 1997. Ants can colour graphs. Journal of the Operational Research Society 48, 295–305.

Deb, K., Anand, A., Joshi, D., 2002. A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation 10 (4), 371–395.

Dorigo, M., 1992. Optimization, Learning and Natural Algorithms (in Italian). Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

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.
Dorigo, M., Stützle, T., 2004. Ant Colony Optimization. MIT Press, Cambridge, MA.
Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1), 29–41.
Dorigo, M., Di Caro, G., Gambardella, L.M., 1999. Ant algorithms for discrete optimization. Artificial Life 5 (2), 137–172.
Dréo, J., Siarry, P., 2002. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In: Dorigo, M., Di Caro, G., Sampels, M. (Eds.), Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002), vol. 2463 of LNCS. Springer-Verlag, Berlin, Germany, pp. 216–221.
Dréo, J., Siarry, P., 2004. Continuous interacting ant colony algorithm based on dense hierarchy. Future Generation Computer Systems 20 (5), 841–856.
Eiben, A.E., Bäck, T., 1997. Empirical investigation of multiparent recombination operators in evolution strategies. Evolutionary Computation 3 (5), 347–365.
Fogel, D.B., Bayer, H.G., 1995. A note on the empirical evaluation of intermediate recombination. Evolutionary Computation 4 (3), 491–495.
Gagné, C., Price, W.L., Gravel, M., 2002. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 53, 895–906.
Gambardella, L.M., Taillard, E., Agazzi, G., 1999. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F. (Eds.), New Ideas in Optimization. McGraw-Hill, UK, pp. 63–76.
Golub, G., van Loan, C., 1989. Matrix Computations, second ed. The Johns Hopkins University Press, Baltimore, MD.
Goss, S., Aron, S., Deneubourg, J.-L., Pasteels, J., 1989. Self-organized shortcuts in the Argentine ant. Naturwissenschaften 76, 579–581.
Guntsch, M., Middendorf, M., 2002. A population based approach for ACO. In: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G. (Eds.), Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTim, vol. 2279 of LNCS. Springer-Verlag, Berlin, Germany, pp. 71–80.
Hansen, N., Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 2 (9), 159–195.
Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer-Verlag, Berlin, Germany.
Kennedy, J., Eberhart, R.C., 1995. Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks, vol. 4. IEEE Press, Piscataway, NJ, pp. 1942–1948.
Kern, S., Müller, S.D., Hansen, N., Büche, D., Očenášek, J., Koumoutsakos, P., 2004. Learning probability distributions in continuous evolutionary algorithms – A comparative review. Natural Computing 3 (1), 77–112.
Mathur, M., Karale, S.B., Priye, S., Jyaraman, V.K., Kulkarni, B.D., 2000. Ant colony approach to continuous function optimization. Industrial and Engineering Chemistry Research 39, 3814–3822.
Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation 6 (4), 333–346.
Monmarché, N., Venturini, G., Slimane, M., 2000. On how *Pachycondyla apicalis* ants suggest a new search algorithm. Future Generation Computer Systems 16, 937–946.
Očenášek, J., Schwarz, J., 2002. Estimation distribution algorithm for mixed continuous-discrete optimization problems. In: Proceedings of the 2nd Euro-International Symposium on Computational Intelligence. IOS Press, Amsterdam, Netherlands, pp. 227–232.
Ostermeier, A., Gawelczyk, A., Hansen, N., 1994. Step-size adaptation based on non-local use of selection information. In: Davidor, Y., Schwefel, H.-P., Männer, R. (Eds.), Parallel Problem Solving from Nature – PPSN III, vol. 866 of LNCS. Springer-Verlag, Berlin, Germany, pp. 189–198.
Pelikan, M., Goldberg, D., Sastry, K., 2000. Bayesian optimization algorithm, decision graphs, and Occam's razor. Technical Report IlliGAL Report No. 2000020, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 2000.
Ralston, A., Rabinowitz, P., 1978. A First Course in Numerical Analysis, second ed. McGraw-Hill, New York, NY.
Reimann, M., Doerner, K., Hartl, R., 2004. D-ants: Savings based ants divide and conquer the vehicle routing problems. Computers and Operations Research 31 (4), 563–591.
Rumelhart, D., Hinton, G., Williams, R., 1986. Learning representations by backpropagation errors. Nature 536, 323–533.
Schwefel, H.-P., 1981. Numerical Optimization of Computer Models. John Wiley and Sons, New York, NY.
Siarry, P., Berthiau, G., Durbin, F., Haussy, J., 1997. Enhanced simulated annealing for globally minimizing functions of many continuous variables. ACM Transactions on Mathematical Software 23 (2), 209–228.
Socha, K., Sampels, M., Manfrin, M., 2003. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Raidl, G. et al. (Eds.), Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization, volume 2611 of LNCS. Springer-Verlag, Berlin, Germany, pp. 334–345.
Storn, R., Price, K., 1995. Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA.
Stützle, T., Dorigo, M., 1999. ACO algorithms for the traveling salesman problem. In: Miettinen, K., Mäkelä, M.M., Neittaanmäki, P., Périaux, J. (Eds.), Evolutionary Algorithms in Engineering and Computer Science. John Wiley and Sons, Chichester, UK, pp. 163–183.
Stützle, T., Hoos, H.H., 2000. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16 (8), 889–914.
Wodrich, M., Bilchev, G., 1997. Cooperative distributed search: The ant's way. Control and Cybernetics 26 (3), 413–446.
Yuan, B., Gallagher, M., 2003. Playing in continuous spaces: Some analysis and extension of population-based incremental learning. In: Sarker, R. et al. (Eds.), Proceedings of Congress of Evolutionary Computation (CEC). IEEE Press, Piscataway, NJ, pp. 443–450.