On how Pachycondyla apicalis ants suggest a new search algorithm

Nicolas Monmarché, Gilles Venturini, Mohamed Slimane

On how *Pachycondyla apicalis* ants suggest a new search algorithm

Nicolas Monmarché, Gilles Venturini and Mohamed Slimane

Laboratoire d'Informatique de l'Université de Tours
Ecole d'Ingénieurs en Informatique pour l'Industrie (E3i)
64, Avenue Jean Portalis, 37200 Tours, France,
monmarche,venturini,slimane@univ-tours.fr
Tel: +33 2 47 36 14 14, Fax: +33 2 47 36 14 22

published in: Future Generation Computer Systems 16 (2000) 937–946

Abstract

In this paper we present a new optimization algorithm based on a model of the foraging behavior of a population of primitive ants (*Pachycondyla apicalis*). These ants are characterized by a relatively simple but efficient strategy for prey search in which individuals hunt alone and try to cover a given area around their nest. The ant colony search behavior consists of a set of parallel local searches on hunting sites with a sensitivity to successful sites. Also, their nest is periodically moved. Accordingly, the proposed algorithm performs parallel random searches in the neighborhood of points called hunting sites. Hunting sites are created in the neighborhood of a point called nest. At constant intervals of time the nest is moved, which corresponds to a restart operator which re-initializes the parallel searches. We have applied this algorithm, called API, to numerical optimization problems with encouraging results.

1 Introduction

Algorithms inspired by models of ant colony behavior are knowing increasing success among researchers in computer science and operations research [1, 6]. One may now find applications of these algorithms in different areas such as robotics [10, 14], objects clustering [3, 16, 17], communication networks [4, 22] and combinatorial optimization [5, 7, 8, 9].

The most successful implementations of ant-inspired algorithms are those for combinatorial optimization and network routing. These implementations [4, 5, 7, 8, 9, 22] share a common structure which recently allowed the definition of a novel metaheuristic called Ant Colony Optimization (ACO) [6]. In ACO algorithms, ants are agents that move on a graph representation of the problem under solution in such a way that the sequence of moves they perform provides a feasible solution to the problem.

In this paper, we are interested in a model of the foraging strategy of the *Pachycondyla apicalis* ponerin ants [11, 12] and in its application to optimization problems. These ants use relatively simple principles to search their preys, both from global and local viewpoints. Starting from their nest, they globally cover a given surface by partitioning it into many hunting sites. Each ant performs a local random exploration of its hunting sites and its site choice is sensitive to the success previously met on the sites. These principles can be used to implement a new strategy for the search of a global minimum of a function f in a search space S.

The remainder of this paper is organized as follows: section 2 describes the main principles used by *Pachycondyla apicalis* while searching preys. Section 3 presents our algorithm called API (after

Pachycondyla API*calis*). Section 4 describes the experimental results that have been obtained on standard numeric tests functions. Section 5 discusses the obtained results and section 6 draws some conclusions and discusses future work.

2 Overview of the foraging strategy of *Pachycondyla apicalis*

Pachycondyla apicalis ants have been studied in the Mexican tropical forest near the Guatemala border [11, 12]. Colonies of these ants comprise approximately from twenty to one hundred individuals; at any given time only a small percentage of them (20%-30%) are hunting.

Their global strategy for prey searching can be characterized as follows. The ants create hunting sites which are distributed relatively uniformly around the nest at a distance of approximately ten meters from it. Hunting sites have a radius of approximately 2.5 m. In this way, using a mosaic of small areas, the ants cover a rather large surface around the nest. Periodically, the nest location changes. These changes can be explained by the fact that the nest may become less comfortable over time, or by prey impoverishment. The nest move is a quite complex process relying on ants specialized in the search of a new site and is based on a recruitment mechanism, called tandem running [15], in which one ant leads another one, and so on.

The local strategy of a foraging ant is as follows. Initially, an ant randomly chooses a hunting site around the nest. Given a first success (prey capture), it memorizes the site. An ant has a tendency to go back to the last successful hunting site using the same path. To follow this path, it uses visual landmarks. When a prey has been captured at a given hunting site, the next exploration performed by the ant always starts from that hunting site. When a hunting site impoverishes and the ant no longer receives the reinforcement represented by the capture of a prey, it has a tendency to explore other hunting sites. More specifically, it can move to a previously explored hunting site, which highlights the ability to memorize several sites. Finally, when a prey is captured, the ant goes straight back to the nest.

Interactions between ants regarding foraging are limited. As mentioned previously, these ants use visual landmarks and not pheromone trails, as many other species of ants do.

3 General algorithmic model of API

In this section we describe our adaptation of the *Pachycondyla apicalis* foraging behavior to the solution of optimization problems. The hunting area of ants corresponds to the problem search space. The foraging strategy of ants corresponds to the search operators used by the optimization method.

3.1 Search space and evaluation function

We consider a population of n ants $a_1, ..., a_n$. These agents are located in a search space S and they try to minimize a function $f : S \to \mathbb{R}$. Each point s in S is a valid solution to the considered problem. S can be a continuous space ($S = \mathbb{R}^\ell$), a binary space ($S = \{0, 1\}^\ell$), or a permutation space (as in the traveling salesman problem). In fact, our algorithm, named API (after *Pachycondyla* API*calis*), does not impose any limitation on the definition of the search space. API requires the definition of only the two following operators:

- $\mathcal{O}_{\text{rand}}$ which generates a random point in S according to a uniform distribution,
- $\mathcal{O}_{\text{explo}}$ which generates a point s' in the neighborhood of a point s.

The size of the neighborhood centered in s is set to a value A, $A \in [0, 1]$. This parameter defines the amplitude of the explorations. When $A = 0$, then s' is always equal to s. When $A = 1$ then s' can be any point in S.

It is important to mention that $\mathcal{O}_{\text{explo}}$ does not have to be a purely random exploration: this operator may also implement a priori known heuristics in the studied domain.

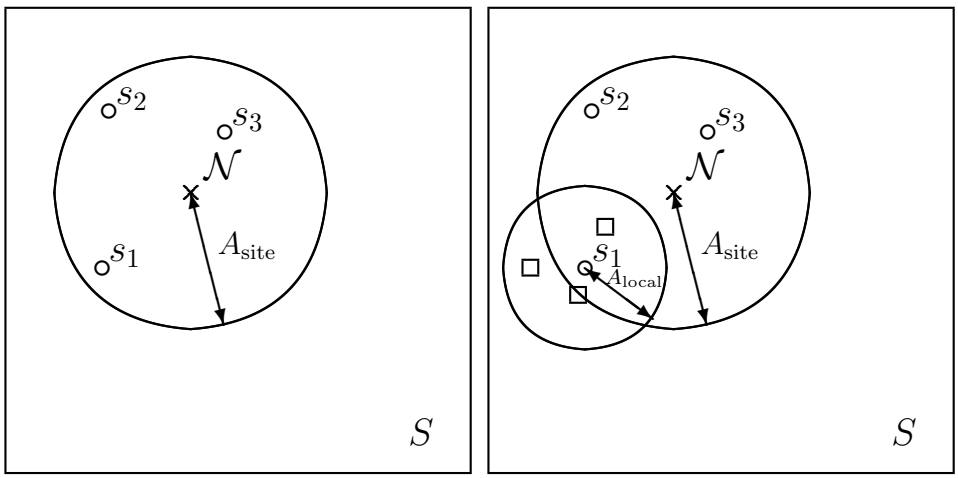

Figure 1: (a) Sites s_1, s_2 and s_3 are randomly generated (their maximum distance from the nest $\mathcal{N}$ is given by A_{site}). (b) Squares represent local explorations of site s_1 at a maximum distance A_{local} from s_1.

3.2 Global exploration of the search space

The API algorithm can informally be described as follows. At the beginning, the nest location $\mathcal{N}$ is placed at a random location determined by $\mathcal{O}_{\text{rand}}$. Then $\mathcal{N}$ is moved each T movements of the n ants[1] and is placed on the best point found since the last nest move. Therefore the displacement of the nest occurs every $n \times T$ individual moves.

When API starts, or after the nest has been moved, each ant a_i leaves the nest to randomly generate p hunting sites in the neighborhood of $\mathcal{N}$. To create these initial hunting sites, each ant a_i makes use of $\mathcal{O}_{\text{explo}}$ with an amplitude A which is set to $A_{\text{site}}(a_i)$ (see Figure 1(a)). The values $A_{\text{site}}(a_i)$ are set as follows:

$$A_{\text{site}}(1) = 0.01, \ldots, A_{\text{site}}(i) = x^i \times 0.01, \ldots, A_{\text{site}}(n) = x^n \times 0.01 = 1$$

where $x = \frac{1}{0.01}^{\frac{1}{n}}$.

Because the number of sites memorized by real ants is not known, we have arbitrarily decided to set the default number of sites memorized in each ant's memory to $p = 2$.

3.3 Local behavior of ants

Each ant a_i locally searches for preys in its hunting sites by performing an exploration of a selected site (see Figure 1.b). Initially, a_i randomly selects one of its p hunting sites and goes there. Let s be this site. Then ant a_i performs a local search in the neighborhood of s: a point s' is selected using $\mathcal{O}_{\text{explo}}$ with an amplitude $A_{\text{local}}(a_i)$ and $f(s')$ is evaluated. A local exploration is successful if it leads to a better value of f, that is, if $f(s') < f(s)$. Then a_i memorizes this success and updates the site s in its memory: $s \leftarrow s'$. In this case, a_i will immediately return to the new site s for its next exploration. If a local exploration of a site is not successful, then in the next exploration a_i will randomly choose a site among its p sites in memory. When a site has been explored successively for more than $P_{\text{local}}(a_i)$ times without catching any prey, it is forgotten and replaced by creating a new site. This new site is created using $\mathcal{O}_{\text{explo}}$ with an amplitude $A_{\text{site}}(a_i)$. Finally, after each nest move, sites are erased from the ants memory (this mechanism is intended to help avoiding local minima). The automaton given in Figure 2 describes the behavior of a single ant. It has been observed in real ants that hunting sites can be located up to 20 meters away from the nest, while the local explorations on a site have a radius

[1] At each step, all ants move in parallel, even if the actual implementation of API is sequential.

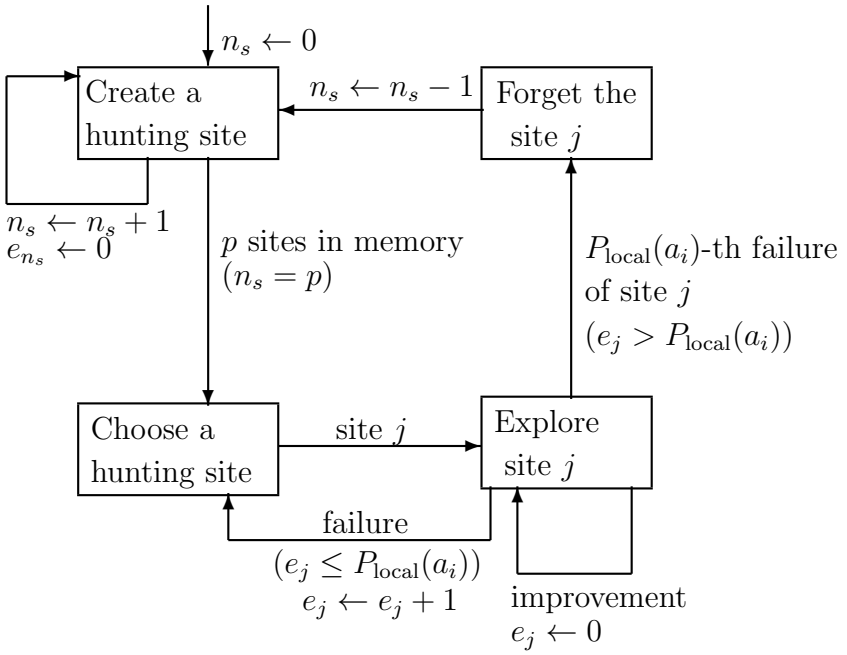

Figure 2: Automaton used to model the behavior of a *Pachycondyla apicalis* ant (a_i). $n_s \leq p$ represents the number of sites memorized by the ant. e_j represents the number of unsuccessful explorations successively performed by the ant on site j.

of up to 2 or 3 meters [12]. Therefore, we have decided to keep this ratio of $\frac{1}{10}$ between long and local range explorations. The amplitude of the local search has therefore been set to $A_{\text{local}}(i) = A_{\text{site}}(i)/10$.

3.4 Cooperation with tandem recruitment

Cooperation in API is not as obvious as in ant based algorithms using pheromone trails (like for instance ACO algorithms [6]). Nevertheless, in API, ants use several forms of implicit or explicit cooperation.

Implicit forms of cooperation in API have been mentioned before, and result from the independent behavior of ants: moving the nest is a form of cooperation in which ants individually search for a future nest location, and after a while share the information collected individually and decide to move it to the most promising location. In the same way, the overall distribution of sites around the nest results from the individual behavior of each ant.

In this section, we describe an explicit form of cooperation called tandem-running (tandem running is a particular form of recruitment, see for example [15]). Each time the n ants explore the nest surroundings, recruitment is performed in the following way: two ants a_i and a_j are randomly selected. Let us suppose that the best site of a_i has a better quality than the best site of a_j. In this case, the best site of a_j will be deleted and replaced by the best site of a_i. This cooperation is a form of exploitation which increases the number of trials that will be allocated to a given site. It also allows API to explore a site with different search parameters because the two ants hunting on the same site may have different parameters.

3.5 Resulting overall algorithm and its main properties

A high level description of the API algorithm is given in Figure 3, while Figure 4 illustrates its behavior graphically. In this last figure, it can be noticed that once a site has been created, it can move step

1. **Choose** randomly the initial nest location $\mathcal{N}$
2. For each ant a_i, $i \in [1 \dots n]$:

   If a_i has less than p hunting sites in memory Then **Create** a new site in the neighborhood of $\mathcal{N}$ and **Explore** this created site

   Else

   If the previous site exploration was successful Then **Explore** again the same site

   Else **Explore** a randomly selected site (among the p sites in memory)
3. **Remove** from the ants memories all sites which have been explored unsuccessfully more than $P_{\text{local}}(a_i)$ consecutive times
4. **Perform** recruitment (best site copying between two randomly selected ants)
5. If more than T iterations have been performed Then **Change** the nest location and **Reset** the memories of all ants
6. Go to (2) or Stop if a stopping criterion is satisfied

Figure 3: The API algorithm.

by step to any place in the search space due to local moves that improve the evaluation function. Therefore, after a while, the distribution of hunting sites may be different from the initial one (see for instance illustrations (b) and (c) in Figure 4).

API's strategy has the following main properties:

- It centers the search around a given point. This is similar to what happens in the delta-coding technique introduced in [23]. In delta coding, the binary representation used in the GA is not the direct representation of a solution but rather the representation of a small displacement (called δ) from a central point (called partial solution). This representation can thus periodically change as the central point moves.
- The nest is moved. This is similar to a restart operator in GAs [18] which consists in (1) stopping the GA before the end of the total run, (2) keeping some information from past generations such as the best individuals, and (3) generating a new population including the kept information. API clearly makes use of a restart operator, and keeps the best point generated since the previous restart.
- Several hunting sites are memorized. Let p be the number of sites that have been created in the ant's memory. The ants' strategy for allocating trials to hunting sites can be analyzed by a Markov chain. If there are p sites and the success probabilities for these sites are p_1, p_2,...,p_p, respectively, then it can be shown that the average number of trials allocated to each site is proportional to its success probability. It means that the ants' strategy is sensitive to success, yet it has a very low computational cost because the success probabilities are not explicitly computed.
- A heterogeneous population is used to avoid critical parameter setting. In many optimization methods, setting good parameter values is difficult because an appropriate choice is often related to the structure of the fitness landscape. With a heterogeneous population, ant parameters may not be optimal for a given problem but will hopefully be robust across a variety of problems. Also, ants with different parameter settings can participate in different search steps (for instance, ants moving in large steps can quickly find solutions that can be further improved by ants moving in small steps).

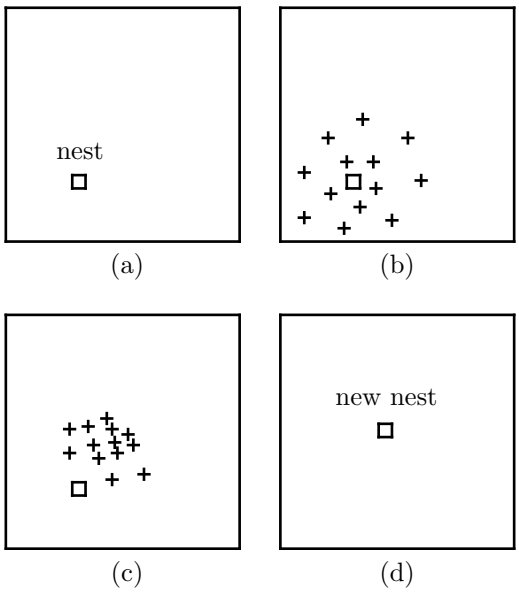

Figure 4: Illustration of API's behavior. In (a), the nest is randomly placed in the search space. Then, in (b), hunting sites are randomly created around the nest with the distribution generated by the A_{site} parameters. In (c), due to the local explorations, hunting sites move towards more interesting areas of the search space (here the center of the space in our example). Then, in (d), the nest moves to the best generated point so far. Hunting sites are then created again as in (b), and so on.

- The parallelization of API on a standard computer network is trivial and most probably efficient. More precisely, a master processor can keep the nest location which is sent to slave processors which perform the local search process of the colony members. Once these processes have ended, slaves send the best individual they have found to the master which updates the new nest location. Communication between processors is thus strongly limited between nest moves, even if recruitment sometimes requires to swap an individual between two processors.

4 Experiments

4.1 Parameter settings

In API there are a number of parameters that need to be set. We set the following default values: the number of ants was set to $n = 20$, the number of explorations performed by each ant between two nest moves was set to $T = 50$. Finally, we set the parameters $P_{\text{local}}(a_i), i = 1, \ldots, n$, to 50. In this way, ants will forget a site only when the nest is moved. Moreover, recruitment is used unless indicated otherwise.

4.2 Experimental settings

We consider here that the search space S is $\mathbb{R}^\ell$ and that the evaluation function $f(x_1, \ldots, x_\ell)$ to be minimized is defined on a subset of $\mathbb{R}^\ell$ delimited by ℓ intervals $[b_i, B_i](\forall i \in [1, \ldots, \ell])$. In this real-coded search space, $\mathcal{O}_{\text{rand}}$ consists in generating a random point within the intervals according to a uniform distribution. For a site $s = (x_1, \ldots, x_\ell)$, the operator $\mathcal{O}_{\text{explo}}$ generates a point $s' = (x'_1, \ldots, x'_\ell)$ as follows:

$$x'_i = x_i + U[-0.5, +0.5] \times A \times (B_i - b_i) \, \forall i \in [1, \ldots, \ell]$$

Table 1: The seven test functions used in this paper. These functions have been used with real or binary encoding and are standard test functions [24]. All these functions have a minimum of 0.

Function		Interval of x_i	x_i encoded on
f_1	$x_1^2 + x_2^2 + x_3^2$	$[-5.12, 5.11]$	10 bits
f_2	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.047]$	12 bits
f_3	$50 + \sum_{i=1}^{5}(x_i^2 - 10\cos(2\pi x_i))$	$[-5.12, 5.11]$	10 bits
f_4	$1 + \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
f_5	$1 + \sum_{i=1}^{5} \frac{x_i^2}{4000} - \prod_{i=1}^{5} \cos(\frac{x_i}{\sqrt{i}})$	$[-5.12, 5.11]$	10 bits
f_6	$0.5 + \frac{\sin^2(\sqrt{x_1^2+x_2^2})-0.5}{1+0.001(x_1^2+x_2^2)}$	$[-100, 100]$	10 bits
f_7	$(x_1^2 + x_2^2)^{0.25}(1 + \sin^2 50(x_1^2 + x_2^2)^{0.1})$	$[-100, 100]$	10 bits

where $U[-0.5, +0.5]$ is a uniform distribution in $[-0.5, +0.5]$, and A is the maximum amplitude of the move.

In Table 1, we give the 7 standard numerical test functions that we used to evaluate API. These functions are defined over a rather low dimensional search space but they have often been used to evaluate other metaheuristics [2, 13, 20, 24].

The basic version of API, which uses the default parameter settings defined in section 4.1, will be denoted API-2s-20a-r (for API with 2 sites, 20 ants and recruitment). We perform now an analysis of the parameter settings on API's performance and in section 4.7 we compare API's results with standard optimization methods.

In the following section, each algorithm run consists in evaluating 10,000 points in S. All presented results are averaged over 50 runs.

4.3 Number of ants

We first studied the influence of the number of ants on API's performance. The results are given in Table 2. The first interesting thing to notice is that API with only one ant is not robust. As mentioned in sections 3.2 and 3.3, A_{site} and A_{local} parameters respectively range from 0.01 to 1 and from 0.001 to 0.1. Therefore, with only one ant, we have two possibilities to set the values of the single ant parameters: ($A_{\text{site}} = 1$, $A_{\text{local}} = 0.1$) or ($A_{\text{site}} = 0.01$, $A_{\text{local}} = 0.001$). Both versions of API with one ant are not robust for the seven tested functions, as highlighted for instance by results obtained on functions f_3 and f_7. Nevertheless, the obtained results are quite good for smooth functions such as f_1 and f_2.

Results obtained with API-2s-2a-r, that is, API with 2 ants (using $A_{\text{site}} = 1$ and $A_{\text{site}} = 0.01$, respectively) are much better than the single ant versions. This shows that cooperation between ants in API is efficient.

Finally, the number of ants does not significantly change API's performance as long as at least two ants can forage. This is especially true for functions f_1, f_2 and f_6. For some functions, such as f_4, f_5 and f_7, there is a small performance decrease. If one ranks the tested versions of API, then API with 20 ants has a mean rank of 2.71 for all functions (see Table 3). This appears to be a good compromise for a parallel implementation: having 20 ants instead of 5 or 10 (both with a rank equal to 2.57) can significantly decrease the necessary time to obtain a similar result on a parallel computer.

4.4 Recruitment and nest moves

We performed one experiment without recruitment, one in which the nest was left at the same position during the whole run and where ants never forgot about their hunting sites ($T = \infty$, $P_{\text{local}} = \infty$),

Table 2: Study of the performance of API as a function of the number of ants (API 2s-30a-r, for instance, stands for API with 2 sites, 30 ants and recruitment). The columns report the average solution quality found over 50 runs with at most 10,000 function evaluations per run. The numbers in parentheses are the standard deviations. Functions f_i are defined in Table 1. Best results are based on 5 digits precision and are in boldface.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7
API 2s-1a $A_{\text{site}} = 1$, $A_{\text{local}} = 0.1$	0.00 (0.00)	0.00 (0.00)	19.05 (4.20)	0.05 (0.02)	1.27 (0.18)	0.01 (0.00)	0.89 (0.25)
API 2s-1a $A_{\text{site}} = 0.01$, $A_{\text{local}} = 0.001$	**0.00** (0.00)	**0.00** (0.00)	28.79 (11.63)	**0.00** (0.00)	**0.09** (0.02)	0.28 (0.17)	7.03 (2.94)
API 2s-2a-r	0.00 (0.00)	0.00 (0.00)	5.06 (3.06)	0.00 (0.00)	0.21 (0.07)	0.00 (0.00)	0.16 (0.05)
API 2s-5a-r	**0.00** (0.00)	**0.00** (0.00)	8.60 (4.28)	0.00 (0.00)	0.15 (0.04)	0.00 (0.00)	**0.12** (0.03)
API 2s-10a-r	**0.00** (0.00)	**0.00** (0.00)	6.83 (3.59)	0.00 (0.00)	0.16 (0.04)	0.00 (0.00)	0.14 (0.03)
API 2s-20a-r	0.00 (0.00)	**0.00** (0.00)	5.26 (2.80)	0.00 (0.00)	0.18 (0.04)	**0.00** (0.00)	0.15 (0.04)
API 2s-30a-r	0.00 (0.00)	**0.00** (0.00)	**4.83** (2.62)	0.00 (0.00)	0.19 (0.06)	0.00 (0.00)	0.16 (0.04)
API 2s-40a-r	0.00 (0.00)	**0.00** (0.00)	5.72 (2.96)	0.00 (0.00)	0.21 (0.06)	0.00 (0.00)	0.17 (0.04)

and finally one experiment with API without nest moves but with ants that forgot about hunting sites ($T = \infty$, $P_{\text{local}} = 50$). In Table 4, the results of these experiments are compared to API using the default parameters settings.

These experiments have shown that:

1. Recruitment has no significant influence on performance except for f_3 which has many local minima and requires more exploration (and thus less recruitment).

2. Nest moves are extremely important to achieve good performance. This is certainly due to the fact that the ants find local minima and need to be redistributed around the central point. In fact, we have observed that a nest move is often followed by an improvement of the best point generated so far.

3. Forgetting the hunting sites seems to be relatively equivalent to nest restart.

4.5 Number of sites

Because the number of explorations is limited for a given nest location, an ant will explore a lot of different areas around the nest and will thus perform more exploration than exploitation if it forages on many sites. On the other hand, with one site only, the ant has no other choice than to concentrate its search effort on the same site. We have tested three values for the number of sites (1, 2 and 5). The results presented in Table 5 suggest that increasing the number of sites leads to a slight decrease in performance.

4.6 Homogeneous versus heterogeneous parameters

Finally, we have tested different ant parameters for local and global search: API-2s-20a-r-ho in which all ants have the same A_{local} ($A_{\text{local}} = 0.1$) and A_{site} ($A_{\text{site}} = 0.5$) parameters (homogeneous parameters), and API-2s-20a-r-uni where $A_{\text{local}}(i)$ is determined according to the distribution adopted all along this

Table 3: Ranks obtained for each function according to the number n of ants. For $n = 1$, we give the ranks for the first two sets of parameters in Table 2.

n	ranks							
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	mean
1	4, 1	3, 1	7, 8	8, 1	8, 1	7, 8	7, 8	10.29
2	3	2	2	2	7	6	5	3.86
5	1	1	6	3	2	4	1	2.57
10	1	1	5	4	3	2	2	2.57
20	2	1	3	5	4	1	3	2.71
30	3	1	1	6	5	5	4	3.57
40	3	1	4	7	6	3	6	4.29

Table 4: Study of the effects that recruitment and nest moves have on API performance. API 2s-20a-norec-nores stands for API with 2 sites, 20 ants, no recruitment and no restart (nest does not move). Best results are based on 5 digits precision and are in boldface.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7
API	0.00	**0.00**	5.26	0.00	**0.18**	**0.00**	0.15
2s-20a-r	(0.00)	(0.00)	(2.80)	(0.00)	(0.04)	(0.00)	(0.04)
API	0.00	**0.00**	**3.55**	0.00	0.18	0.00	0.19
2s-20a-norec	(0.00)	(0.00)	(1.92)	(0.00)	(0.06)	(0.00)	(0.05)
API 2s-20a-norec-nores	**0.00**	0.00	10.72	0.02	0.37	0.00	0.42
$P_{\text{local}} = \infty$, $T = \infty$	(0.00)	(0.00)	(4.94)	(0.01)	(0.10)	(0.00)	(0.19)
API 2s-20a-r-nores	**0.00**	**0.00**	8.12	**0.00**	0.19	0.00	**0.09**
$P_{\text{local}} = 50$, $T = \infty$	(0.00)	(0.00)	(4.66)	(0.00)	(0.07)	(0.00)	(0.02)

paper (i.e., $A_{\text{local}}(1) = 0.001, \ldots, A_{\text{local}}(n) = 0.1$) and the A_{site} parameter is set to 0.5 (in this case, the initial distribution of hunting sites around the nest is globally uniform). As can be seen from the obtained results (see Table 6), homogeneous parameters do not allow API to be well adapted to the different functions, while heterogeneous parameters do (API 2s-20a-r outperforms the two other algorithms on all functions). With API 2s-20a-r-uni, ants do not concentrate their search around the nest but rather do a lot of exploration. This is the main reason explaining the relatively poor results obtained by that variant.

4.7 Comparative study

To evaluate the performance of API, we have ran comparisons with the following algorithms:

- A random hill climbing algorithm (RHC). Initially, RHC randomly generates a point s in S. Then it iteratively generates a point s' in the neighborhood of s and replaces s with s' if $f(s') < f(s)$.
- A multiple restart random hill climbing algorithm (MRRHC). MRRHC is the same as RHC but when the random search is not improving anymore, that is, if the best value is not improved after a given number R of iterations, then it is restarted from a randomly generated point.
- A generational genetic algorithm (GA) with binary tournament selection, 1 point crossover and with a real-coded representation [19].

The same $\mathcal{O}_{\text{explo}}$ operator has been used for performing mutations in the tested GA and random exploration in the RHC and MRRHC methods. Thus all methods use the same search operator.[2]

[2]We could have used a gradient-like operator in API, but this would not have been fair to the other methods.

Table 5: Study of the effects of the number of sites on API performance. Tested values are $p = 1, 2$ and 5 sites. Best results are based on 5 digits precision and are in boldface.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7
API	**0.00**	**0.00**	6.87	**0.00**	**0.14**	0.00	**0.12**
1s-20a-r	(0.00)	(0.00)	(4.81)	(0.00)	(0.04)	(0.00)	(0.03)
API	0.00	**0.00**	**5.26**	0.00	0.18	**0.00**	0.15
2s-20a-r	(0.00)	(0.00)	(2.80)	(0.00)	(0.04)	(0.00)	(0.04)
API	0.00	0.00	7.40	0.00	0.23	0.00	0.20
5s-20a-r	(0.00)	(0.00)	(3.78)	(0.00)	(0.06)	(0.00)	(0.06)

Table 6: Results obtained with API when all ants have homogeneous parameters (API 2s-20a-r-ho) and when the global distribution of hunting sites is uniform around the nest (API 2s-20a-r-uni, where the A_{site} parameters are all equal). Best results are based on 5 digits precision and are in boldface.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7
API	**0.00**	**0.00**	**5.26**	**0.00**	**0.18**	**0.00**	**0.15**
2s-20a-r	(0.00)	(0.00)	(2.80)	(0.00)	(0.04)	(0.00)	(0.04)
API	0.00	0.00	11.59	0.06	1.17	0.00	0.71
2s-20a-r-ho	0.00	0.00	2.89	0.02	0.15	0.00	0.19
API	0.00	0.00	7.73	0.02	1.10	0.00	0.35
2s-20a-r-uni	(0.00)	(0.00)	(3.43)	(0.01)	(0.22)	(0.00)	(0.13)

We have tested RHC, MRRHC and GA with different sets of parameters. The final parameter settings have been determined by first optimizing the parameters on each function separately and then averaging the parameters over the 7 functions (the parameter values are reported in the first column of Table 7).

Comparative results are presented in Table 7. API outperforms the GA on functions f_2, f_4, f_6 and f_7. f_2 is part of De Jong's test suite and is known as being difficult for GAs. The GA outperforms API on function f_3. This can be explained by the fact that function f_3 requires a local search amplitude which is higher than 10%, the maximum local amplitude used in API. In fact, the experiments performed when testing the parameters of the GA, the RHC and the MRRHC, suggest the use of an amplitude of 10-20% in the local search when dealing with f_3.

RHC or MRRHC are definitely not competitive with API (but sometimes outperform the GA, as on functions f_2 and f_6).

5 Discussion

The conclusions below can be drawn from the tests we have performed, but the reader should keep in mind that these conclusions are limited to the tested problems.

The number of ants in API does not seem to be a critical parameter in the case of a sequential implementation, as long as there are enough ants to ensure the two following properties:

1. A minimal exploration takes place to ensure the efficiency of the nest move.
2. The number of ants must be large enough in order to ensure that the ant's parameters are heterogeneous. This is because the distribution of the A_{site} and A_{local} parameters depends on the colony size.

The number of 20 ants, as suggested in section 4.3, seems to be a good value.

Table 7: Average results obtained on the seven numerical test functions. Number in parentheses are standard deviations. The parameters of the GA were $P_{\text{cross}} = 0.8$ and $P_{\text{mut}} = 0.1$. A is the amplitude used by RHC, MRRHC and GA in the $\mathcal{O}_{\text{explo}}$ operator. R is the maximum number of unsuccessful explorations performed by MRRHC after which the search is restarted, $|Pop|$ is the population size in the GA. Best results are based on 5 digits precision and are in boldface.

Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7
API	0.00	**0.00**	5.26	**0.00**	0.18	**0.00**	**0.15**
2s-20a-r	(0.00)	(0.00)	(2.80)	(0.00)	(0.04)	(0.00)	(0.04)
RHC	0.00	0.00	7.08	0.03	0.82	0.00	0.54
$A = 10\%$	(0.00)	(0.00)	(3.27)	(0.01)	(0.15)	(0.00)	(0.13)
MRRHC	0.00	0.00	14.89	0.20	1.77	0.04	0.74
$A = 1\%$, $R = 40$	(0.00)	(0.01)	(5.67)	(0.16)	(1.46)	(0.02)	(0.53)
GA $A = 8\%$	**0.00**	0.04	**2.54**	0.02	**0.14**	0.06	0.43
$\|Pop\| = 100$	(0.00)	(0.05)	(1.58)	(0.03)	(0.07)	(0.05)	(0.88)

Recruitment and the number of sites are not critical with respect to performance. But two properties of API seem to be crucial:

1. Moving the nest and performing several restarts during the search, and
2. using an heterogeneous population of ants.

In addition, the overall distribution of hunting sites around the nest seems to be quite important.

API can also be applied to numerical optimization problems using a binary encoding. In this case, $\mathcal{O}_{\text{rand}}$ would randomly generate a binary string of length ℓ and $\mathcal{O}_{\text{explo}}$ would flip between 1 and $A \cdot \ell$ bits of a given binary string. Combinatorial problems can also be tackled. For instance, in the traveling salesman problem, the search space S corresponds to all possible tours, that is, all possible permutations of cities. $\mathcal{O}_{\text{rand}}$ would generate a feasible tour and $\mathcal{O}_{\text{explo}}$ would use standard operators like the 2-opt operator or the city-insertion operator [21].

Preliminary tests of API on different problems in numerical and combinatorial optimization seem to indicate that API can achieve robust performance for all the tested problems.

6 Conclusion

In this paper, we have proposed a new search algorithm inspired by the behavior of *Pachycondyla apicalis* ants. This algorithm has many features such as multiple local search processes centered around multiple points, a restart operator, a strategy sensitive to the success of evaluated points and a heterogeneous population of ants. It is a general optimization algorithm that can be applied to continuous and discrete optimization problems.

Results have shown that the important characteristics of API are a combination of a restart operator, of a heterogeneous population of ants, and of the use of hunting sites from which the search is started. Experimental studies suggest that API may achieve interesting results.

Among the possible perspectives, we are currently testing API on other domains in numerical optimization with constraints, such as the learning of hidden Markov models for pattern recognition problems. In addition, we are studying how to parallelize API on a standard computer network in order to tackle large size problems.

Acknowledgement

We would like to thank the anonymous reviewers, our colleague C. Duprat and the editors for their valuable help and comments on this paper.

References

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

[2] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[3] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamic of collective sorting robot-like ants and ant-like robots. In J.A Meyer and S. Wilson, editors, *First International Conference on Simulation of Adaptive Behavior*, pages 356–365. MIT Press, Cambridge, Massachusetts, 1991.

[4] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998. (http://www.jair.org).

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

[6] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, UK, 1999.

[7] M. Dorigo and L.M. Gambardella. Ant Colony Sytem: A cooperative learning approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

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

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

[10] A. Drogoul, J. Ferber, B. Corbara, and D. Fresneau. A behavioral simulation model for the study of emergent social structures. In F. Varela and P. Bourgine, editors, *First European Conference on Artificial Life, Paris, France*, pages 161–170. MIT Press, Cambridge, Massachusetts, 1991.

[11] D. Fresneau. Individual foraging and path fidelity in a ponerine ant. *Insectes Sociaux, Paris*, 32(2):109–116, 1985.

[12] D. Fresneau. *Biologie et comportement social d'une fourmi ponérine néotropicale (Pachycondyla apicalis)*. PhD thesis, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, France, 1994. Thèse d'Etat.

[13] V.S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann, San Mateo, California, 1993.

[14] S. Goss and J.-L. Deneubourg. Harvesting by a group of robots. In F. Varela and P. Bourgine, editors, *First European Conference on Artificial Life, Paris, France*, pages 195–204. MIT Press, Cambridge, Massachusetts, 1991.

[15] B. Hölldobler and E.O. Wilson. *The Ants*. Springer Verlag, Berlin, Germany, 1990.

[16] P. Kuntz and D. Snyers. Emergent colonization and graph partitioning. In D. Cliff, P. Husbands, J.A. Meyer, and Stewart W., editors, *Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 494–500. MIT Press, Cambridge, Massachusetts, 1994.

[17] E.D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In D. Cliff, P. Husbands, J.A. Meyer, and Stewart W., editors, *Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 501–508. MIT Press, Cambridge, Massachusetts, 1994.

[18] J. Maresky, Y. Davidor, D. Gitler, G. Aharoni, and A. Barak. Selectively destructive re-start. In L. Eshelman, editor, *Sixth International Conference on Genetic Algorithms*, pages 144–150. Morgan Kaufmann, San Francisco, California, 1995.

[19] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, third edition, 1996.

[20] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R.K. Belew and L.B. Booker, editors, *Fourth International Conference on Genetic Algorithms*, pages 271–278. Morgan Kaufmann, San Francisco, California, 1991.

[21] G. Reinelt. The traveling salesman, computational solutions for TSP applications. In *Lecture Notes in Computer Science*, volume 840. Springer Verlag, Heidelberg, Germany, 1994.

[22] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1997.

[23] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: an iterative search strategy for genetic algorithms. In R.K. Belew and L.B. Booker, editors, *Fourth International Conference on Genetic Algorithms*, pages 77–84. Morgan Kaufmann, San Francisco, California, 1991.

[24] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In L. Eshelman, editor, *Sixth International Conference on Genetic Algorithms*, pages 239–246. Morgan Kaufmann, San Francisco, California, 1995.