Noname manuscript No.
(will be inserted by the editor)

Revisiting wavefront construction with collective agents: an approach to foraging

Olivier Simonin · François Charpillet · Eric Thierry

Received: date / Accepted: date

Abstract We consider the problem of coordinating a team of agents that have to collect disseminated resources in an unknown environment. We are interested in approaches in which agents collectively explore the environment and build paths between home and resources. The originality of our approach is to simultaneously build an artificial potential field (APF) around the agents' home while foraging. We propose a multi-agent model defining a distributed and asynchronous version of Barraquand *et al.*'s Wavefront algorithm. Agents need only to mark and read integers locally on a grid, that is, their environment. We prove that the construction converges to the optimal APF. This allows the definition of a complete parameter-free foraging algorithm, called c-marking agents. The algorithm is evaluated by simulation while varying the foraging settings. Then we compare our approach to a pheromone-based algorithm. Finally, we discuss requirements for implementation in robotics.

Keywords Swarm algorithm · Foraging task · Artificial Potential Field · Ants

O. Simonin
INSA Lyon, CITI-Inria Lab., France
and Inria Nancy Grand Est, LORIA Lab., Maia team (until 30 August 2013)
Tel.: +33-4-72436422
Fax: +33-4-72436227
E-mail: olivier.simonin@insa-lyon.fr

F. Charpillet
Inria Nancy Grand Est, LORIA Lab., Maia team, France
Tel.: +33-3-83593043
Fax: +33-3-83278319
E-mail: francois.charpillet@loria.fr

E. Thierry
ENS Lyon, France
Tel.: +33-4-26233933
Fax: +33-4-72728080
E-mail: eric.thierry@ens-lyon.fr

1 Introduction

Swarm intelligence is now recognized as a robust and flexible paradigm to deal with distributed problems in complex environments. Among the approaches, the use of synthetic pheromones is one of the most popular ways to address problems such as foraging (Parunak, 1997; Panait and Luke, 2004), path-planning (Parunak et al., 2002; Sauter et al., 2002) and multi-agent patrolling (Sempe and Drogoul, 2003; Sauter et al., 2005).

In this paper we focus on the foraging problem, which is a well-known testbed for multi-agent and multi-robot systems (Steels, 1989; Winfield, 2009). In the foraging problem, several agents search an unknown environment for objects/food sources, and eventually transport them to their home(s).

The foraging activity is particularly challenging when the environment is unknown (no map) and complex (e.g. containing concave obstacles). It concerns both physical environments and graphs (e.g. Wagner et al. 1998).

We are interested in approaches in which agents/robots have limited abilities, little or no memory and are potentially numerous. In these approaches, the environment is used by agents for their coordination. It is assumed that agents can read and write information in the environment. This is a well-known approach in living systems, for instance ants that leave pheromones to build paths in foraging problems (Bonabeau et al., 1999). Ants inspired several pheromone-based algorithms for search problems. They rely either on computing diffusion and evaporation of digital pheromone deposits (Deneubourg et al., 1986; Resnick, 1994) or updating timestamps on visited nodes of a graph (e.g. Wagner et al. 1998; Glad et al. 2008). However, ant models have some drawbacks such as requiring the fine tuning of the free parameters (evaporation and diffusion rate) and the risk of building local minima in the pheromone field where ants/agents can get trapped. Moreover, some models require expensive computation such as diffusion of one or several pheromones.

In this paper we propose a foraging approach that does not depend on free parameters and does not require expensive computation as with pheromone-based models. We aim at defining an efficient way for the homing task (finding a path towards home) that avoids local minima, whatever the topology of the environment.

Our approach is inspired by centralized planning in which the environment is known. We refer to popular techniques based on Artificial Potential Field (APF) approaches (Khatib, 1985; Arkin, 1998). The APF approach consists in computing and adding repulsive potential fields from obstacles and an attractive field to the goal (as functions of the distances, see (Khatib, 1985)). It allows a robot to go downhill toward the goal by following the negative gradient of the composite potential. However, it is well known that the composite potential may contain minima where robots can be trapped (see Zhu et al. 2006). The APF construction proposed by Barraquand *et al.* (1992) is promising in this regard since it prevents the formation of local minima. It consists of computing one field from the goal, following a BFS[1]-like algorithm on a grid, so giving a shortest path from any cell. In order to apply this approach in an unknown environment, we translate the algorithm into the collective behavior of simple autonomous agents.

[1] Breadth First Search

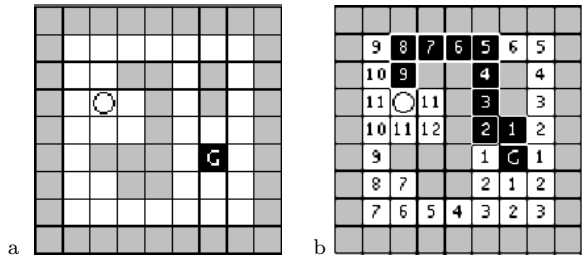

Fig. 1 WaveFront algorithm: a) initial environment state and b) result of the construction. Gray cells are obstacles and white cells are free cells. Cells are colored in black to represent the shortest path from the agent, i.e. the circle, to the goal, i.e. to the 'G' cell.

In this paper we define an asynchronous computation of the Barraquand *et al.* wavefront based on memoryless agents marking their environment. We prove its convergence to the optimal APF. Then we extend the approach to define a new foraging algorithm relying on the computation and the use of this APF building. As the APF values are persistent in the environment, we can define deterministic behaviors that allow the marking and following of paths between resources and the home, without requiring any parameter. We give proofs of convergence and termination of these behaviors, before studying their performances in simulation and comparing them to a classical ant model.

The rest of the paper is organized as follows. In Section 2, we present the Barraquand *et al.* wavefront algorithm building a numerical potential field for path-planning. In Section 3, we define a memory-less multi-agent algorithm that builds a numerical potential field converging to the wavefront field. We prove this convergence. In Section 4, the agent behavior is used to define a first foraging algorithm, whose termination is studied. In Section 5, we extend the approach to define the c-marking algorithm. In Section 6 we analyze performance of this algorithm in simulations. Then we compare to the performance of a pheromone-based (ant) algorithm in Section 7. We discuss related work in Section 8 and a possible robotic implementation in Section 9. Finally, Section 10 concludes the paper.

2 Barraquand *et al.* wavefront computation

We first examine the pre-computed map proposed by Barraquand and colleagues (1991; 1992) which provides, from any place in a given discrete environment, a shortest path toward a goal. Local minima are avoided by computing a "wavefront expansion" *from the agent/robot destination*, which involves building an ascending APF incrementally.

The environment is represented as a 2D-array of cells and is denoted by E (see Figure 1.a). E_{empty} denotes the subset of E containing all obstacle-free cells. Each cell $c = (x, y) \in E$ has a maximum of four reachable neighbors:
$(x-1, y), (x+1, y), (x, y-1)$ and $(x, y+1)$.
A potential field is a function $V : c \in E_{empty} \longmapsto V(c) \in \mathbb{R}$.
Let $c_{goal} \in E_{empty}$ be the goal cell of the robot. The wave-potential V is computed as shown in Algorithm 1 where $V(c_{goal})$ is set to 0 and L_i denotes a list of cells

Algorithm 1 WAVEFRONT EXPANSION (Barraquand et al., 1992)

For every cell $c \in E_{empty}$, set $V(c)$ to infinity (e.g., a large number M)
Set $V(c_{goal})$ to 0 and the list L_0 to (c_{goal}), $i = 0$
While L_i is not empty Do
  initialize L_{i+1} to the empty list
  For every cell c in L_i
    For every neighbor n of c in E_{empty}
      if $V(n) = M$ then set $V(n)$ to $i + 1$ and insert n at the end of L_{i+1}
  $i = i + 1$

in E (i.e., those having distance i from c_{goal}). At each iteration of the main loop, L_{i+1} is computed and i is increased. The algorithm terminates when E_{empty} has been totally explored. Figure 1.b presents the result of the algorithm execution on a test environment where the goal cell is located at letter G. Each cell has a potential value equating to the minimum distance to the goal. We remark that this algorithm is the classic BFS algorithm computed on a grid world (which is a graph). So it requires robots to have a discretized map of the environment to compute it.

Using this computed field, an agent can now reach the goal by following the flow of the negative gradient. In a numerical potential field, such a behavior is called a "descent" and is defined as follows: at every iteration, the agent examines the potential values of the four neighboring cells and moves to the cell with the smallest value (ties are broken arbitrarily). In Figure 1.b the black cells represent a descent path. An agent occupies a cell and it is represented by a circle.
More formally, the following two properties derive from Algorithm 1 (cf. details in (Barraquand et al., 1992)):

Property 1: In the numerical potential field produced by Algorithm 1, an agent that always moves to the neighboring cell with the smallest potential value arrives at the goal location and succeeds in avoiding all obstacles.

Property 2 : From any location in the numerical potential field (Alg. 1), the path induced by Property 1 is the shortest.

The first step of our proposal is to define a *multi-agent* algorithm that builds a numerical potential field such as the one produced by Algorithm 1 and that holds these two properties.

3 Computation with memory-less agents

We study how the wavefront computation of Barraquand *et al.* (1992) can be achieved on-line by a set of autonomous agents evolving on a grid. The main property is that agents do not need an *a priori* knowledge of the world and do not need to know their global position in the grid.

The considered agents are memory-less, that is, they do not have individual memory but they are able to cooperate through indirect communication. This form of communication is embedded in the environment. It is inspired by ants that drop pheromones as markings to be read later by others (including possibly themselves), see Holldobler and Wilson (1990). The environment allows the aggregation of the local interactions of numerous agents in order to build collective patterns

Algorithm 2 EXPLORATION & APF CONSTRUCTION (around cell c_0)

EXPLORATION
IF there exist neighboring cells ***without a value*** THEN
  move to one of them (random selection) and call UPDATE-VALUE
ELSE randomly move to a neighboring cell and call UPDATE-VALUE

UPDATE-VALUE
Compute $val = 1 + min$(4-neighbor values)
Write val in the current cell if this cell is different from c_0

(Bonabeau et al., 1999). The environment becomes a shared memory in which information can be stored and removed (according to a sequential access).

Thus we have the central idea of our first algorithm: numerical values of the APF to be built are treated as markings that can be laid down and updated by agents.

3.1 Simple "marking agents"

Let us now describe how agents asynchronously build the APF, that is, without building it by incremental distance. The environment is a grid with free places and obstacles defined by occupied places. The size of the cells corresponds to the agent size. Free places can store an integer value which is the value of the APF for the considered location. We consider that agents move following the iterations of an infinite loop (or for a fixed time). At each iteration, all agents move sequentially to a neighboring cell and write a value. Each agent computes a piece of the APF while discovering the environment.

Agents do not know their global position, but rather occupy a cell. Initially, the value of the goal cell c_0 is set to 0, this is the single cell whose value cannot change. All the agents are initially placed on cell c_0 (agents can start from any position but the construction is then longer). Without loss of generality and for the sake of simplicity, we allow several agents to occupy a cell at the same time. Each agent is capable of:

- reading and writing an *integer* value in the cell it occupies
- perceiving the four neighboring cells: detecting the cells which are obstacles or reading their values
- moving to a neighboring cell that is not an obstacle.

We call such agents *marking agents*.

In order to build incrementally the wavefront around the initial value (the goal cell), each agent repeatedly follows Algorithm 2 (EXPLORATION & APF CONSTRUCTION). Agents move preferentially to not yet explored cells, otherwise they choose randomly an obstacle-free cell with a uniform distribution. Then they perform the UPDATE-VALUE operation which corresponds to a local wavefront expansion in the Barraquand *et al.* algorithm. As a consequence, the shortest path from this new cell to the goal cell, in the current APF, is equal to *the shortest distance to a neighboring cell + 1 (one move)*. Figure 2 illustrates the incremental building of the APF.

As agents favor *unmarked* cells over marked ones (see the first line of Algorithm 2) their moving is not just random exploration. Up to a certain distance, their

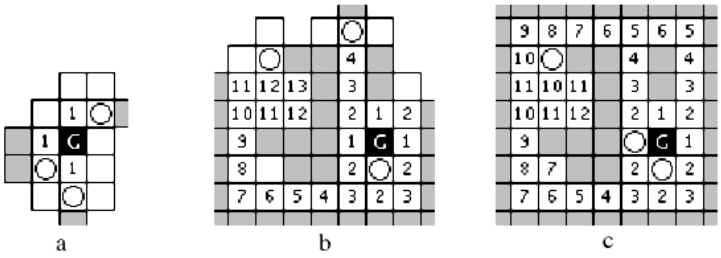

Fig. 2 Collective construction of the potential field with 3 agents (iterations 2, 17 and 65).

spatial progression corresponds to the cells of the front computed by the centralized algorithm. However, the number of agents is generally not sufficient to sustain all the cells corresponding to the wavefront at the same time (this is the case in Fig. 2.a after the first iteration).

As a consequence, one exploration iteration, when all the agents move, is not equivalent to one loop of Algorithm 1. Figure 2.b shows a situation where the cell with value 13 has clearly not reached the value of the shortest distance to G (which is 11, see Figure 2.c). Agents may therefore need to visit the same cell several times before it reaches its optimal value. As the environment is bounded, the time necessary to obtain the complete optimal potential field is finite (with probability 1). The algorithm convergence is proved below.

3.2 Convergence

Let us consider the following notations:

- $v(c)$=current value of already visited cell c ($v(c_0)$ is initialized to 0)
- $v^*(c)$=optimal value of already visited cell c (distance to c_0)
- $L_i = \{c \mid v(c) = i\}$ (L_i contains all cells having currently distance i)
- $L_i^* = \{c \mid v^*(c) = i\}$ (L_i^* contains all cells having their optimal distance i)

Remark: $\forall t,\ \forall c,\ v(c) \geq v^*(c)$. An immediate consequence is that, at any time, a cell $c \in L_i^*$ is part of some set L_j with $j \geq i$.

Before proving the convergence of algorithm 2, we need to consider its ability to visit and revisit all the cells of the environment. We study this preliminary property in the following lemmas 3.1 and 3.2.

Lemma 3.1. *Considering a set of agents performing Algorithm 2 in a bounded environment, each obstacle-free cell will be visited in finite time with probability 1.*

Proof: Let $S_{visited}$ be the set of already visited cells, $S_{boundary}$ the set of visited cells having one non-visited cell among its neighbors and S_{inside} the complementary set $S_{inside} = S_{visited} \backslash S_{boundary}$.

When an agent is in S_{inside}, it performs a random walk within $S_{visited}$ which consists in moving to a neighbor cell with uniform probability. We are interested in the time to hit $S_{boundary}$ (from which the agent will explore a new cell). To show that it is finite with probability 1, consider this random walk over all $S_{visited}$

cells (assuming that the walk does not stop at $S_{boundary}$ but continues within $S_{visited}$ with the same rule). It is a discrete time-homogeneous Markov chain over the cells $S_{visited}$. This set of states is closed and finite, thus recurrent (J. Norris's book 1998, Theorem 1.5.6) and, since it is irreducible, the time to hit any cell in $S_{boundary}$ from any starting cell is finite with probability 1 (Norris 1998, Theorem 1.5.7). The time to reach such a cell can be arbitrarily long, however (we evaluate later the consequences of using random walks).

When the agent reaches a cell of $S_{boundary}$, it moves to a non-visited cell, whose state switches to *visited*, and which then belongs to S_{inside} or $S_{boundary}$. The number of random walks is bounded by the number of cells because each random walk ends with the addition of a new cell to $S_{visited}$. So, each agent alternates a finite number of times between finite length random walks and moves extending $S_{visited}$. This process stops when all cells are in $S_{visited}$, which happens in finite time (with probability 1). □

LEMMA 3.2. *Once each cell has been visited once, the time between two visits of a given cell is finite with probability 1.*

Once all cells have been visited, $S_{visited}$ can no longer change and from that time the moves of an agent follow exactly the random walk described in the previous lemma. It is a discrete time-homogeneous Markov chain over $S_{visited}$. The set of states is closed and finite, thus recurrent (Norris 1998, Theorem 1.5.6) and, since it is irreducible, the time to hit any cell from any starting cell is finite with probability 1 (Norris 1998, Theorem 1.5.7). □

Same remark as above: the time for each revisit can be arbitrarily long.

We can now consider the **convergence** property of algorithm 2:

THEOREM 3.3. *In a bounded discrete environment, a set of agents performing Algorithm 2 builds the optimal APF in finite time with probability 1.*

Proof: Let us prove by induction that, for all i, we have $L_i = L_i^*$ in finite time (with probability 1).

This is true for L_0 as $v(c_0) = v^*(c_0) = 0$ in the initial state and no other cell has a value lower or equal to 0.

Let us now consider that this is true for some $i = 0, \ldots, k-1$. $L_k \backslash L_k^*$ is a finite set and each cell of $L_k \backslash L_k^*$ will be visited or revisited in finite time (Lemma 3.2). When a cell $c \in L_k \backslash L_k^*$ is visited, its value $v(c)$ is updated to $v^*(c) = k$ because at least one of its neighbors is in L_{k-1}^*. Indeed, the maximum gradient difference between two neighboring cells in the optimal APF is one (and L_{k-1}^* is built). As a result, we obtain $L_k = L_k^*$ in finite time.

Because there is a finite number of sets L_k^* (bounded by the number of cells), Algorithm 2 builds the optimal APF in finite time with probability 1. □

Note that while the participation of multiple agents speeds up convergence (see Section 6), these properties are valid also when only one agent is used. However, the APF construction can be long, due to the random walk used by agents when they evolve in already visited regions. In practice, using several agents allows the avoidance of long convergence times (see study of scalability in Section 6.4). Ant algorithms also use a random walk strategy for exploration, we will compare to such a model in Section 7.

The next section shows how the proposed multi-agent construction can be used to define a foraging algorithm.

4 Extension to collaborative foraging

4.1 The multi-source foraging problem

Foraging is a benchmark for multi-agent and multi-robot systems. It was inspired by the foraging task of some ant species, which search for food in unknown environments before transporting it to their nest (homing), see (Steels, 1989).

There are many variants of this problem based on assumptions about robots and the environment (Winfield, 2009). However, we want to focus on the two main tasks, searching and homing, while considering simple agents/robots. So, in the rest of the paper we consider that agents can detect and manipulate resources in the environment, and have no energy limits. However, we consider finite search spaces, a classic setting that represents energy limitation (Ferber, 1999).

Let us note that several round-trips may be necessary to completely upload all resources at a given place. Therefore task completion requires the agents to be able to return from the base to a discovered resource location. The environment may and usually does contain obstacles. The difficulty of foraging depends on the number and shape of the obstacles. We do not consider dynamic environments (e.g., moving sources) that define other complex variants of the problem.

In practice, the assumption that the robots forage in an unknown environment is often relaxed by making some other assumptions. In particular, many models consider that the base emits a signal allowing agents to perceive its direction in order to come back (simplifying the problem and its implementation in robots), see for instance (Drogoul and Ferber, 1992). However, even such facilities cannot help the robots navigate through obstacles, especially if they are complex (e.g. with concave shapes). In the proposed model, we do not consider such external facilities or knowledge of the environment.

4.2 Foraging using marking agents

Let us consider environments that have resources in multiple locations. These locations are unknown to the agents, they contain a given quantity of resource which is also unknown. We consider, in the rest of the article, that a cell of the environment can contain:

- an *obstacle* or wall (cell colored in Gray),
- a *resource* (dark Gray) whose quantity q_i is bounded,
- the *base* ('B' letter in black cell), where the resources have to be brought,
- one or several *agents* (whose ID number is shown in a circle in the cell).

All agents start from the base (there is no limit to the number of agents per cell). In addition to the abilities mentioned in section 3.1, an agent can also

- detect the presence of resources in the four neighboring cells,
- load a quantity q_{max} of resources.

Building a wavefront from the base is useful for any task that requires to come back to the base. Whatever its position in the environment, an agent only has to descend the numerical gradient in order to reach the base. Obstacles are not a problem. We prove in the next subsection that, even if being asynchronously built,

the wavefront under construction remains monotone, that is, an agent can never fall into local minima.

To abstract, the proposed approach is a marking agent with the ability to return to the base as often as required. We define this re-entrant behavior as a descent motion (presented in Section 2). Algorithm 3 represents the complete behavior of such an agent, repeating sequentially the tasks SEARCH and RETURN TO BASE.

Figure 3 provides a schematic overview of this foraging approach.

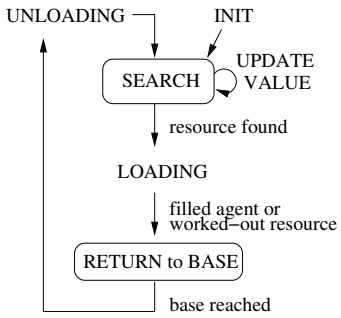

Fig. 3 Foraging with a marking agent.

Algorithm 3 SEARCH & RETURN TO BASE

SEARCH (Repeat)
IF a ***resource*** *is detected* in a neighboring cell THEN
move into this cell, load a quantity q_{max} of resource and execute RETURN to BASE
ELSE execute EXPLORATION & APF CONSTRUCTION (i.e. Alg. 2)

RETURN TO BASE (Repeat)
IF agent reaches the ***base*** THEN unload resource and execute SEARCH
ELSE *move* to a neighboring cell ***that has the smallest value*** and call UPDATE-VALUE

Note that this multi-agent system can perform three tasks simultaneously :

- *exploration* to discover resources,
- *construction* of an optimal APF,
- *transportation* of resources to the base (homing).

Figure 4 illustrates the progression of this foraging algorithm in a small maze containing resources at five locations (4.a) and the final state obtained after 500 iterations (4.b) showing the optimal APF.

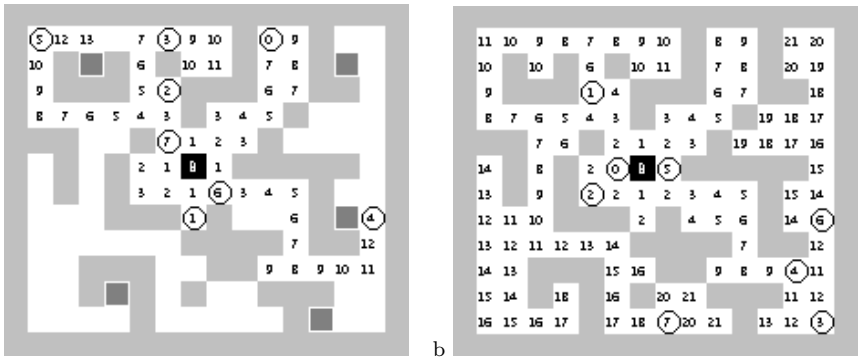

Fig. 4 Foraging with 8 marking agents in a small maze (a) after 18 iterations (b) in final state. The base is the black center cell, resources are the 5 dark gray cells.

4.3 Soundness of the APF construction

Contrary to the Barraquand *et al.* algorithm and as stated in Section 3.1, the value of a cell is not necessarily optimal after its first visit, but it converges monotonically to the optimum. It is a consequence of the random nature of the agents' exploration, contrary to its incremental computation in algorithm 1.

Thus, one may wonder if the agents can get trapped in local minima when returning to the base. The answer to this question is: no. To prove this we first consider the following lemma:

LEMMA 4.1. *During the APF construction, any cell other than c_0 which is revisited can only change for a smaller value.*

Proof: Consider any reachable cell c_i different from c_0, and c_{min} its neighboring cell having the minimum value[2]. We have $v(c_i) = 1 + v(c_{min})$ because the UPDATE operation is done at every visit of cell c_i.

Suppose now that $v(c_i)$ changes for a greater value when applying the UPDATE operation. This implies that $v(c_{min})$ has changed previously to a greater value. So the same reasoning must be applied to c_{min} for which the minimum neighboring cell must have been increased before, and so on, until reaching c_0 as the minimum neighboring cell. But $v(c_0) = 0$ and cannot be changed, making the hypothesis absurd. The UPDATE operation can only compute a value lower than or equal to the current one. □

THEOREM 4.2. *During the APF construction, for each reached cell c other than c_0, there exists at least one neighboring cell with a value lower than $v(c)$.*

Proof: Consider any reachable cell c_i different from c_0, and c_{min} its neighboring cell having the minimum value. We examine if visits to the c_i neighboring cells could make c_i a local minimum. We consider two cases :

(a) A neighboring cell of $c_i \neq c_{min}$ receives a new value. If this value is greater than or equal to $v(c_{min})$ then $v(c_i) = 1 + v(c_{min})$ is unchanged. If the value is lower than $v(c_{min})$ then c_i takes a new neighboring minimum, noted $c_{min'}$, which verifies the property $v(c_{min'}) < v(c_i)$.

(b) If the c_{min} neighboring cell is visited, Lemma 4.1 ensures its value can only change for a lower value, so still verifying $v(c_{min}) < v(c_i)$.

The visit of any c_i neighboring cell cannot prevent c_i from holding a lower-value neighboring cell. As each neighbor calculation is independent, the property is verified for any simultaneous visit of the neighboring cells. □

Theorem 4.2 permits any agent that evolves in the APF under construction to perform a descent to the base in a finite time, since from each cell there always exists a lower-value neighbor to move to. As a consequence, at any point of the exploration, from any cell, our algorithm provides a valid path for any agent to return to the base, whatever the number of obstacles and their shape.

This result has consequences for the termination of the foraging.

[2] If several neighboring cells have the minimum value, c_{min} is one of them.

4.4 Termination of algorithm 3

Obviously a first consequence of Theorem 4.2 is that task RETURN TO BASE *terminates*. Indeed, RETURN TO BASE is computed as a descent to the base when a resource is discovered (visited).

A second consequence is that the foraging terminates (Algorithm 3). We consider that the foraging is finished when all resource locations are exhausted. We show that our algorithm will reach this state if the environment is finite and contains finite resources. As quantity of resource is bounded, a resource location will be exhausted in a finite number of visits. So, to demonstrate that the foraging terminates, we must verify that all resource locations will be discovered and revisited until exhaustion. The properties of visiting and revisiting all cells have been proved in Section 3 with Algorithm 2 (EXPLORATION & APF CONSTRUCTION), in Lemmas 3.1 and 3.2. Now consider the question with Algorithm 3.

Algorithm 3 consists of repeating successively the SEARCH and RETURN TO BASE tasks, as the latter terminates. Initially agents execute the SEARCH task. At least one of them will terminate if there are resources in the environment. It is due to the fact that SEARCH consists of executing Algorithm 2, which was proved to visit all cells in finite time (Lemma 3.1).

When an agent finds resources it executes RETURN TO BASE, which does not change the set of visited cells. So its execution has no consequences on the validity of Lemmas 3.1 and 3.2.

When agents have unloaded their resources at the base, they restart SEARCH behavior (i.e. Algorithm 2). As RETURN TO BASE terminates, agents will always return to the SEARCH behavior.

As long as there remain resources in the environment, at least one SEARCH will terminate in finite time by visiting or revisiting a cell with resources (due to Lemmas 3.1 and 3.2). Then the quantity of resource decreases to 0 in finite time, in other words, the foraging of Algorithm 3 terminates. □

We conclude by showing the *property of convergence to the optimal APF* of Algorithm 3 (i.e. Theorem 3.3 of Algorithm 2). We have seen that Lemmas 3.1 and 3.2 are unchanged in Algorithm 3. For the proof of Theorem 3.3, we need to show that executing RETURN TO BASE cannot change the value of the cells that have converged. This is true because RETURN TO BASE can only compute the UPDATE-VALUE operation. □

Note that the foraging can end before the APF has converged.

5 c-marking agents: colorings the paths

5.1 Algorithms

Let us now reconsider the foraging problem in order to allow agents to efficiently return to already discovered resources.

In the previous section, we saw that once a location containing a resource is discovered, the task RETURN TO BASE makes the agent follow a valid path back to the base. The idea is to make *the agent mark the trail while going to the base.*

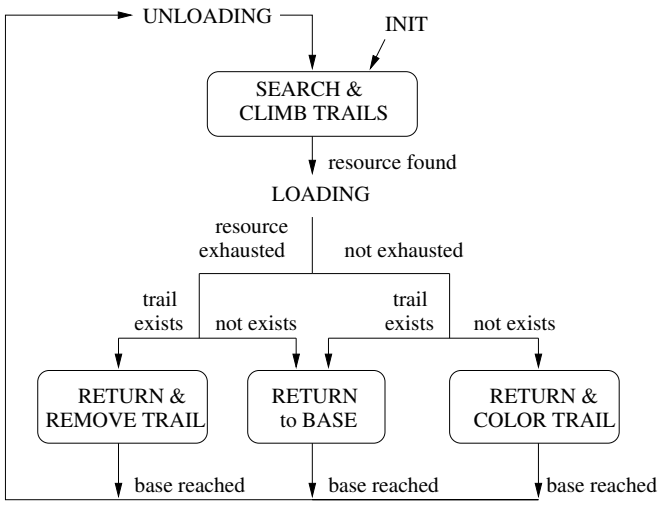

Fig. 5 A c-marking agent colors paths between resource locations and the base.

Then returning to discovered resource locations will be as simple as following this trail.

We now detail the whole approach, called c-marking agent, presented in Algorithm 4 and represented in Figure 5.

RETURN & COLOR TRAIL task. In order to allow agents to mark trails in the numerical potential field, they have the capability to color visited cells. Each trail corresponds to a specific color, which is a function of the agent ID and the time of the resource discovery[3]. As several trails can overlap, a cell can have several colors. This differentiates our approach from existing models that use one or two pheromone markers. However, our approach could use a single color for all the trails. In this case, agents must be able to detect trails crossing when they write or remove a trail (this version is not developed in this paper).

For an agent, creating a trail from a new discovered resource location consists in performing its return to the base while adding the trail color in the visited cells. This task terminates as the trail is built by descending the field to the base (it is a finite path, see Theorem 4.2).

SEARCH & CLIMB TRAIL task. When an agent crosses a colored trail it may follow it to reach a resource location. It is important to show that this task terminates. As values of trail cells can change since they were colored, agents could be blocked in a local maximum. This problem is avoided as *agents follow a trail by moving to the next cell of the same color, that is, independently of the cell's value.* Values are considered only to enter the trail: the agent determines the direction to follow by considering the highest neighboring colored value. Even if cell values where the agent enters have changed, the agent only risks going toward the base and it will not become blocked.

Note that a trail marking in a static APF cannot generate a colored region (e.g., an area of 2×2 cells), as a shortest path is followed to return to the base.

[3] RGB = agent ID (8 bit) + #iteration (modulo 2^{16}) (16 bit)

Algorithm 4 COLOR-MARKING AGENT

SEARCH & CLIMB TRAIL (Repeat)
IF a resource is detected in a neighboring cell THEN move into that cell and exec. LOADING
IF there is no neighboring cell, other than previous position, with a color THEN
  execute EXPLORATION & APF CONSTRUCTION /*i.e. Alg. 2*/
ELSE
  IF previous position was not a trail cell
  THEN Move to the highest-valued colored neighboring cell (1)(2) /*entering trail*/
  ELSE Move to a new cell with the current trail color (1) /*following trail*/
UPDATE-VALUE

LOADING
Pick up a quantity q_{max} of resource
IF the cell is not exhausted of resources yet THEN
  IF the cell is colored **(it is part of a trail)**
  THEN execute RETURN TO BASE /*i.e. Alg. 3*/
  ELSE execute RETURN & COLOR TRAIL
ELSE execute RETURN & REMOVE TRAIL

RETURN & COLOR TRAIL (Repeat)
Color the cell in ***a specific color***, UPDATE-VALUE
IF the agent is located at ***the base*** THEN
  unload resource and execute SEARCH & CLIMB TRAIL
ELSE
  Move to a neighboring cell with the ***smallest value***

RETURN & REMOVE TRAIL (Repeat)
IF the cell has the trail color THEN Remove ***this color***, UPDATE-VALUE
IF the agent is located at ***the base*** THEN
  unload resource and execute SEARCH & CLIMB TRAIL
ELSE
  IF there is at least one neighboring cell with the trail color THEN
  Move to a neighboring trail cell (the smallest one if several are available)
  ELSE Move to the smallest neighboring cell

(1) do random selection if several neighbors are available.
(2) do random selection if several colors are available.

However, such a region could appear, with a low probability[4], if the APF is under construction (i.e. if some cells can still be updated). That is why the climbing behavior chooses randomly a colored neighboring cell, different from the previous position, if several are available (see Alg. 4 /*following trails*/ footnote (1)). This allows an agent to escape such possible small regions with a random walk strategy in order to continue to follow the trail.

RETURN & REMOVE TRAIL task. When a resource location becomes exhausted, the trail leading to it must be removed as quickly as possible, since now it is of no use to any agent. In a standard ant model, the pheromone evaporation mechanism automatically removes useless trails. In our evaporation-free approach, this task is done by the module RETURN & REMOVE TRAIL, in which the agent follows the colored trail while removing this color from the visited cells. If a part of the

[4] Consider the marking of a 3-cell long trail in a 2×2 area (i.e. the trail is turning). If during the two last iterations, the not yet marked cell of the 2×2 area is updated, the trail could continue to this cell, then forming a 2×2 region. Such a region was observed only in simulations with a high density of agents (i.e., number of agents is greater than 25% of the size environment).

colored trail forms a region, at least a 2×2 cells area, the agent could arrive on a cell without colored neighboring cells, what we call a dead end[5]. In this case, the agent simply goes down the APF. The process terminates when the agent reaches the base.

The LOADING task can release three different return tasks. Note that trails are only used to go back to discovered resource locations. Returning to the base is unchanged, as descending the APF allows to perform a shortest path in the current APF.

It is important to note that this version of the proposed approach keeps the properties studied in the previous sections. *Convergence* to the optimal APF is unchanged as the coloring processes can only change a cell's value by calling UPDATE-VALUE. *Termination of the algorithm* is a consequence of the termination of the behaviors Coloring, Climbing and Removing trails (see above). As in the previous version (Algorithm 3), if the resources are bounded, a finite number of SEARCH-RETURN behaviors will be required to exploit all sources. Then the algorithm terminates in finite time.

5.2 Illustration of foraging with c-marking agents

We use the TurtleKit simulation platform (Michel et al., 2005) (part of the multi-agent MadKit Platform (Gutknecht and Ferber, 2001)) for testing our algorithm. In this platform, all agents are activated sequentially at each iteration in random order.

We test c-marking agents within rectangular environments (grids of various sizes), in which obstacles are defined by occupied cells following a desired density. These cells are randomly chosen with a uniform distribution.

The remainder of this section illustrates the whole behavior of the system by describing an experiment with the following first scenario.

Setup 1 (a):

- A 40 cell x 40 cell environment, where 30 % of the cells (randomly distributed) are obstacles. 20 cells are randomly distributed resource locations. Each resource location contains 10 units of resource.
- The number of agents is 10. Each agent can carry a maximum of one unit.

Agents are initially located at the base, which is located at the center of the environment. We observe in Figure 6.a that the construction of the numerical potential field starts out as a partial WaveFront expansion (iteration 11). We can also observe that agent 9 discovers a resource location and starts to create a trail back to the base. Figure 6.b shows that at iteration 21 two other agents discover resource locations and start to create trails, while agent 9 performs multiple trips from the base to the discovered resource location. Later on, at iteration 53, Figure 6.c shows that six resource locations are discovered, with many agents involved in transporting resources along trails marked by others. Note that an agent may follow a trail even if it does not connect to the base at that point in time (e.g.,

[5] To reach such a "dead end" is also necessary that the values of the area have changed since their coloring

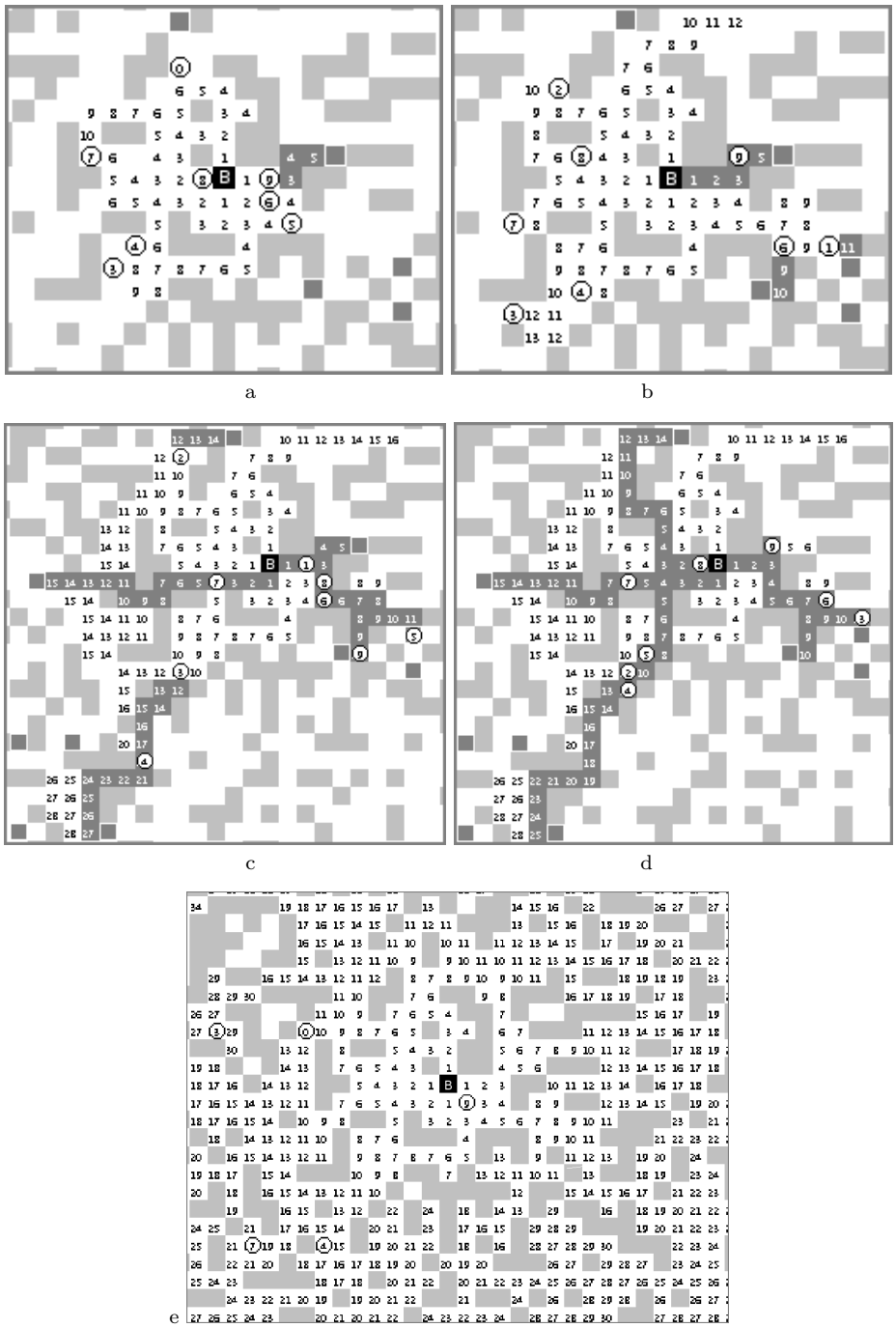

Fig. 6 Resolution of the foraging problem with 10 c-marking agents (with **Setup 1 (a)**). Trails are represented in dark Gray color. Snapshots at iteration 11(a), 21(b), 53(c), 76(d) zoom around the base. The snapshot (e), at iteration 1200, shows the final state of the environment.

agent 4 Fig.6.c). This communication through trails permits the recovery of all the resources in the environment while the APF construction proceeds in parallel. Figure 6.d shows iteration 76, where agents 2, 4 and 5 work simultaneously along the same trail (connecting to a resource location at the bottom of the snapshot). The trail connecting the first discovered resource location is now removed (by agent 9) after being exhausted.

Note that when agents leave the base, they may have to choose among several trails. They randomly choose one, whatever the number of discovered resource locations.

At around the 1200^{th} iteration, all resource locations in the environment have been discovered and thereby exhausted and, additionally, the potential field has converged to its optimal value (Figure 6.e). A video of such a foraging can be found in Online Resource 1. The next section is devoted to quantitatively evaluating algorithm performances.

6 Evaluation of the algorithm (c-marking agents)

6.1 Methodology

Criteria — We now study the performance of the c-marking algorithm. We measure the foraging time, which is the number of iterations required by the agents to discover and exhaust all the resources in the environment. To evaluate the average foraging time for a given setup, we repeat foraging runs until the performance stabilizes. The random number seed is changed at each run. We denote X_n this average time, where n is the number of agents.

We also consider the global *cost* C_n to achieve the foraging with n agents. C_n is computed as $n{\cdot}X_n$. So $n{\cdot}X_n$ stands for the total distance covered by all agents.

Basic environment — Environments are squares in all experiments. All the agents start their exploration from the base which is located at the centre. Obstacles and resource locations are distributed randomly in the environment.

6.2 Performance study

Let us first describe performances in a representative scenario, called **Setup 1 (b)**. This one is based on **Setup 1 (a)** (defined in section 5.2), where the density of obstacles is decreased to 20% in order to reduce unreachable resources (i.e. surrounded by obstacles).

Figure 7.a shows the performance of the algorithm as the number of agents grows. The algorithm was first evaluated with five agents and then this number was progressively doubled to reach 160. Each average foraging time was obtained from 1000 runs.

As Figure 7.a shows, foraging becomes faster as the number of agents increases. Up to 20 agents, with each doubling of agents, the number of iterations required for foraging is halved or reduced even further. This shows the effect of cooperation between agents. The standard deviation can be large, as there is a variation between the environment configurations (obstacles and resources location are generated randomly). However, it is interesting to note that the standard deviation

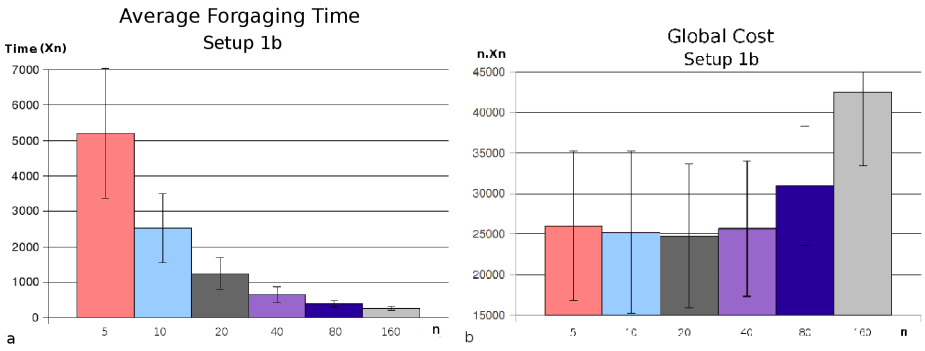

Fig. 7 Performance of the c-marking algorithm in **Setup 1 (b)** (20 resource locations and 20% obstacle density). a) Average time to achieve foraging (X_n). b) Global cost of foraging ($n{\cdot}X_n$). Error bars are the standard deviation.

decreases when the number of agents increases, see fig. 7.a. This is a consequence of the agent spatial dispersion that allows an efficient environment coverage, whatever the location of obstacles and resources. On the contrary, when the size of the environment increases so does the standard deviation, as we will see in subsection 6.4 (Table 1).

We also evaluated the algorithm by varying other parameters such as obstacle density, resource location density, the quantity of resource at each location, in addition to the environment size and the number of agents. For all test configuration, we obtained the same performance profile as that of figure 7.a. At the beginning of the curve, when using few agents, the foraging time decreases dramatically. Then, as the number of agents grows, the curve asymptotically converges to the minimum time required. This results from the minimum amount of time required (i) to reach the resource locations and (ii) to carry all the resources to the base (depends on minimum distance between resource locations and the base).

The performance of the algorithm raises a number of questions. What is the effect of cooperation on performance? How does it scale up with the environment size? How does our algorithm compare to a pheromone-based ant algorithm? In the next sections we attempt to answer these questions.

6.3 Superlinear performance

To study the benefit of cooperation between agents, we analyze the global cost C_n when the number of agents increases.

Figure 7.b plots $n{\cdot}X_n$ values for **Setup 1 (b)**. We can see that up to 20 agents the cost decreases as the number of agents increases. This shows cooperation results in superlinear performance. Beyond this value, the cost increases as the number of agents grows.

Although increasing the number of agents decreases the foraging time, this does not necessarily mean superlinear performance. Figure 8 illustrates that the cost can have different profiles, without including necessarily superlinear performances.

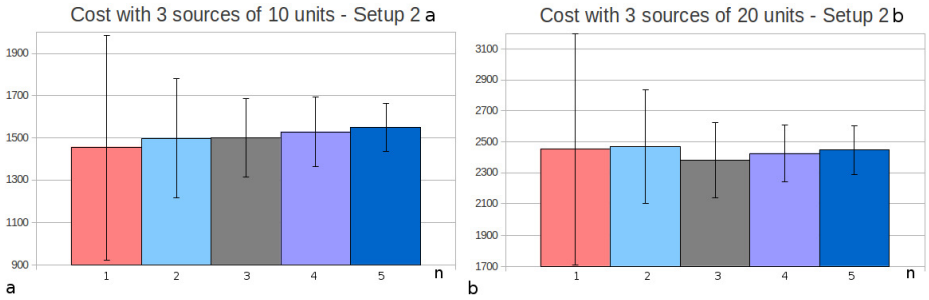

Fig. 8 Global cost with **Setup 2** when varying the number of agents : three resource locations containing 10 units (a) and 20 units (b). (2000 runs for each average time).

This illustration uses **Setup 2**, which contains three sources. We consider two different quantities of resource in the sources.

Setup 2 (a) and (b):

- A 20 cell x 20 cell environment, where 5% of the cells are obstacles. There are three randomly located resource locations each containing (a) 10 units (b) 20 units of resource.
- The number of agents ranges from 1-5. Each agent can carry a maximum of one unit.

We can see in figure 8.a that no superlinear performances are obtained with three resource locations of 10 units. When the quantity of resources is doubled (see fig. 8.b) it appears that the optimal cost is obtained with three agents, a superlinear performance compared to one and two agents.

This last result leads to a question concerning the main factors that affect performance. A foraging process is composed of exploration phases (to find resource locations) and exploitation phases (to transport resources). It is clear that increasing the quantity of resources will increase the exploitation phase (i.e. the number of round-trips required to exhaust the resource locations). Cooperation between agents occurs when they simultaneously explore the environment. Conversely, a single agent will forage by repeating a sequence of search and transport phases. The difference will be significant if the transport phases last long enough, it is true when resource locations contain enough resource. Figures 8.a and 8.b illustrate this difference.

6.4 Influence of the size of the environment (scalability)

In this section we examine how the size of the environment affects the performance of the algorithm when the number of agents is fixed. Obviously, extending the environment will require more time for exploration and transportation of resources to the base.

To conduct this analysis, we computed the performances with **Setup 3** when we progressively quadruple the size of the environment.

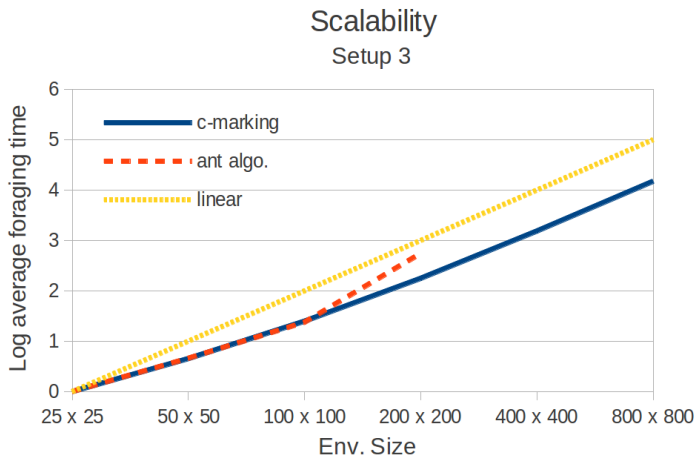

Fig. 9 Scalability of c-marking agents (blue curve) and ants (red curve) when the size of the environment is varied - quadrupled at each step - in **Setup 3**.

Setup 3:

- The environment size is progressively quadrupled from 25×25 to 800×800.
- 5% of the cells are obstacles. 20 cells are randomly distributed resource locations, each containing 20 units of resource.
- The number of agents is 50. Each agent can carry a maximum of one unit.

Average performance times are given in Table 1.

Table 1 Influence of the size of the environment on the performance of c-marking agents using **Setup 3**.

c-marking perf.	25x25	50x50	100x100	200x200	400x400	800x800
average time	341	847	2369	7691	28294	111265
std. deviation	41.6	139.8	590	3196	11313	46408

To examine the scalability of the approach, we plot the time on a logarithmic scale, following the environment size variation. For a series of time $y_0, y_1, .., y_n$ we plot $y = log_4(y_i/y_0)$. The corresponding curve is presented in Figure 9 (the blue curve). The yellow curve shows a linear reference. We can see that times are proportional in the growth of the environment, up to 800×800.

This illustrates that the approach has an efficient scaling. However, as the exploration behavior of the agents uses a random walk, the time follows an exponential growth for larger environments (runs were too long to compute times with size 1600×1600). In the next section, we compare the model with an ant algorithm and discuss the effect of random walk.

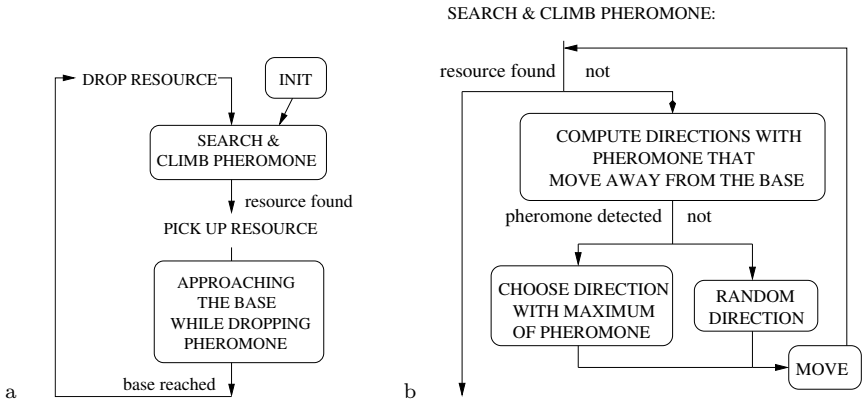

Fig. 10 The ant algorithm: (a) global ant behavior (b) details of task SEARCH & CLIMB PHEROMONE.

7 Comparison with a pheromone-based model

7.1 The ant algorithm

We compare the performance of our algorithm to an ant algorithm based on digital pheromones. In nature, ants deposit a chemical substance called pheromone in the area in which they explore (Holldobler and Wilson, 1990). In computer models these deposits are made when ants find food and return to their nest, in order to create paths from food locations to the nest (Resnick, 1994; Parunak, 1997). Diffusion allows the enlargement of paths in order to recruit other ants. Evaporation deletes useless paths from exhausted resources.

We have implemented such an algorithm, presented in Figure 10, to deal with the foraging problem considered in this article. The default behavior is searching for resource, detailed in Figure 10.b. It consists in climbing trails of pheromone if this allows to move away from the base, otherwise a random walk is performed until perceiving pheromone or finding resources. In this latter case, the ant behavior consists in moving towards the base while dropping pheromones (see Fig. 10.a).

This behavior requires an external information which is the base direction. Indeed, in a model using only one pheromone, ants need information about base location for homing and exploration. The base direction is used to (i) follow pheromone trails in the opposite direction from the base, in order to return to discovered resources, and (ii) to go back to the base. Moreover, this algorithm requires

- each ant is able to drop an amount of pheromone in the current cell (the quantity of pheromone in a cell i, j at time t is denoted $q_{i,j}(t)$)
- each ant can move towards four different neighboring cells and perceive them to detect obstacles and to read the quantities of pheromone
- each ant knows the direction of the base from its current cell
- the environment computes pheromone diffusion (rate α, $0 < \alpha < 1$):
  $\forall i, j \; q_{i,j}(t+1) = (1-\alpha)q_{i,j}(t) + \frac{\alpha}{4}(q_{i-1,j}(t) + q_{i+1,j}(t) + q_{i,j-1}(t) + q_{i,j+1}(t))$
- the environment computes pheromone evaporation (rate ρ, $0 < \rho < 1$):
  $\forall i, j \; q_{i,j}(t+1) = \rho \cdot q_{i,j}(t)$.

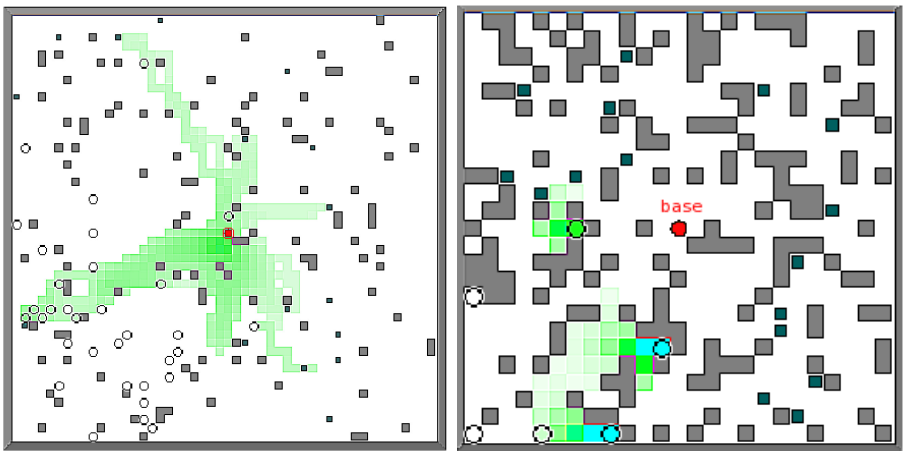

Fig. 11 Foraging with ants that drop pheromones and know the base direction (50×50 env., 50 ants, 20 resources, base at the center). a) obstacle density of 5% does not interfere with foraging b) obstacle density of 30% generates cavities where ants get trapped (zoom $\times 2$). Dark-gray cells are obstacles and circles are the agents. The green, or light gray, color intensity represents the pheromone amount in cells.

The model has been implemented in a Java simulator (see a video in Online Resource 2). The random walk computes a new direction with probability 0.1. The amount of pheromone dropped per cell is +10 and its detection threshold is 1.

Simulations with the ant algorithm (see Figure 11.a) show that ants collectively build pheromone trails between the discovered resources and their base, and follow them to exhaust the resources.

We have observed that ants can get trapped in cavities formed by just a few obstacle-cells (Figure 11.b). Even for a density as low as 5 %, such obstacles may be randomly generated. The problem can be partially solved by adding a random walk in the homing behavior allowing to escape some cavities.

7.2 Measures and comparison

We evaluated the performance of the ant algorithm in **Setup 3**. This requires tuning the free parameters of the ant algorithm for each configuration, that is for each size of the environment. These parameters are the diffusion rate α and the evaporation rate ρ.

Optimizing parameters of a multi-agent system or a bio-inspired metaheuristics may be complex as the sensitivity to parameter variation may be great. To fine tune the parameters of the ant algorithm, we implemented the iterated racing algorithm proposed by López-Ibáñez *et al.* (2011).

After tuning the parameters for each environment size using **Setup 3** (25×25 up to 200×200), we computed the average time to complete the foraging in each setting. The results are presented in Table 2. The table contains the average time and the standard deviation for each configuration. We also give the ρ and α parameters obtained from the iterated racing algorithm.

It is important to note that the time to optimize the parameters was greater than the time to compute average performances. Moreover, we were forced to limit the tuning to 200×200 environments as the time with 400×400 was too long. It is the consequence of the exploration process, based on a random walk, that time is exponential with the size of the environment.

Setup 3 contains less than 5% obstacles to avoid ants becoming blocked by obstacles, allowing the comparison of performance with the c-marking algorithm. We repeat the results with c-marking agents in the last row of Table 2. In all different configurations, we can see that the c-marking algorithm takes on average less time to finish the foraging. The average time for ants is about 40% greater than c-marking agents until the size of the environment reaches 100×100, at which point the time difference rises to 180%. The scalability of ants is nevertheless good up to 200×200 as shown in Figure 9 by the red curve.

Table 2 Performances of the ant algorithm after tuning the free parameters (given in the table) and the c-marking algorithm, by varying the size of the environment in **Setup 3**.

	25x25	50x50	100x100	200x200
ants (av. time)	481	1204	3243	21576
std. deviation	78.6	197	1153	26180
ρ (evap)	0.16	0.05	0.02	0.0005
α (diff.)	0.4	0.1	0.01	0.005
c-marking (av. time)	341	847	2369	7691

It is interesting to examine why the c-marking algorithm has better performance. The difference stems from the ability to explore the environment and to return to a discovered source. Concerning exploration, c-marking agents combine a random walk with a priority to non-visited cells while ants use only a random walk (when they do not detect pheromones). Consequently, c-marking agents reduce the time taken to perform a complete exploration of the environment. Another difference is the ability of c-marking agents to build a persistent trail once a resource location is discovered and until it is exhausted. At the opposite, in the ant algorithm, a pheromone trail could evaporate if no other ant revisits the resource location early enough, which is a probabilistic behavior. However, the persistence of trails in the c-marking agents could have a cost. The trails do not necessarily provide a shortest path from the base to a resource location (this arises during the APF construction). In the ant approach, pheromone evaporation allows to adapt and to optimize the trails. This is a very interesting feature for dynamic environments.

7.3 Discussion of model requirements

When considering the two approaches, the ant algorithm presents different requirements and abilities compared to the c-marking algorithm.

All marking-based approaches require some environment mechanisms. This is the case for our approach (reading and writing integer values) as well as for the ant algorithm. However, the ant approach requires some additional computational

costs. Even if the evaporation process can be performed by agents when they visit a cell[6], pheromone diffusion involves a computational expense within the environment (note that this is not the case with real pheromones). The computation of diffusion and evaporation is not required in the c-marking algorithm. More significantly, our approach has no free parameters to tune before being used efficiently, contrary to the ant algorithm for which the tuning is costly (see previous section).

Furthermore, the ant algorithm requires that agents know/perceive the base direction. This external information is used to follow pheromone trails and to return home. However, this information is generally not sufficient to reach the base when the environment contains complex shaped obstacles. As mentioned above, we observed that ants can get trapped in cavities. To address this problem some extensions of the model have been proposed in the literature. The gist of these attempts is that generally a secondary pheromone is spread from the base in order to build a new gradient centered on the base (e.g. Panait and Luke 2004; Hoff et al. 2010).

The comparison we have proposed is, however, limited to foraging in static environments. We plan to continue this work by considering unbounded environments and dynamic scenarios (as moving obstacles and sources). Pheromone-based models are known to adapt to dynamic environments, as the evaporation process allows a continuous update of the field (Parunak, 1997; Winfield, 2009). Note that our APF computation can also adapt to obstacle changes, as the UPDATE operation is always executed at each cell visit. Concerning the management of trails, our algorithm will require to be adapted, in order to compare with ants.

The differences between the two approaches are real. On one side, the ant algorithm is a bio-inspired model, taking advantage of the adaptive nature of the pheromone. On the other side, we explore an approach which is based on path planning engineering. As a consequence, our approach is limited to some specific tasks and presents few adaptive abilities. For instance, it does not deal with regulation of recruitment. The difference between the two approaches relies mainly on the nature of the marks, each of which has varying abilities. Continuation of this work is motivated by the challenge of bringing closer the approaches.

8 Related work

8.1 A(ge)nts algorithms

Wagner and colleagues (1998; 2000) proposed ant algorithms to deal with search problems on bounded graphs. Their work focuses on the covering task, which consists of visiting all nodes of the graph, using one or several simple marking agents. The proposed algorithms do not deal with all tasks of the foraging problem, such as homing. However, the algorithm by Wagner *et al.* relies on writing integer values on edges or vertexes of the graph, the computation of which is close to our artificial potential field (APF) construction. It uses the same distance update operation, 1 + min(neighboring values), but provides a very different result, as explained below.

[6] It requires that agents write both the date t_v and the relative quantity of pheromone q_v in visited cells, then when reading a cell $q(t) = \rho^{t-t_v}.q_v$.

In Wagner's algorithm, agents build a gradient from their initial position, while all cells are initially set to 0. Ants simply descend this gradient to explore the environment. In our algorithm we set only one cell to value 0, the goal, *which cannot change*. Then the computed APF converges to a static positive gradient built around the goal. Conversely, Wagner's approach builds a dynamic APF as every cell can continually increase in value. As a consequence ants descend toward unvisited cells, performing an efficient search.

Note that our algorithm is sub-optimal for exploration as it uses a pseudo-random walk. We plan to extend our approach by coupling it with an efficient exploration algorithm.

8.2 Gradient computation in grid environments

As expressed in Section 7, pheromone-dependent methods need ad-hoc mechanisms to be applied when the environment contains obstacles and the agents are required to find a path to the base/nest. This can be compass information or external information (e.g. Vaughan et al. 2000).

This problem has motivated a number of variants for computing pheromone gradients. One of these variants is the model of Panait and Luke (2004), which combines pheromones with reinforcement learning as it builds numerical gradients. This approach is meant to provide collaborative foraging, optionally with mobile resources (preys). In this work, "ants" have abilities relatively similar to those of our marking agents. They can move, perceive neighboring cells and read/write real values. Each agent updates the amount of pheromone when visiting a cell by using a reinforcement learning update rule (such as TD(λ) learning). Repeating this update operation leads to the diffusion of values (rewards) set at home and at discovered resource locations. Although this principle requires numerous visits to the cells – or numerous agents – , it is close to the wavefront expansion exploited in the c-marking algorithm. However, the model of Panait and Luke (2004) does not guarantee the avoidance of building local minima.

More generally, other learning-based approaches have been proposed to take advantage of reinforcement or evolutionary algorithms. Agents read the amount of pheromone to update the so-called Q-values (state-action utility estimates) in order to improve exploration or the gradient ascent (e.g. Monekosso et al. 2002). Genetic algorithms are then used to fine tune different parameters and to optimize agent policies (exploration versus exploitation ratio), as in (Sauter et al., 2002). It is worth pointing out that these learning algorithms require additional time, both to compute the propagation and evaporation dynamics and to fully converge to the optimal policies.

Our approach for building an APF in the environment can be seen as a form of incremental search algorithm for path planning (to the base). In particular, we can compare to the D* algorithm (Stentz, 1995; Koenig and Likhachev, 2005) which builds paths from the target and which is able to correct the node costs as the environment is discovered[7]. However, the changes are propagated as waves in the environment representation, while we let the agents update the values when

[7] These algorithms are dedicated to real-time re-planning in dynamic and unknown environments.

Fig. 12 Picture of the apartment in the LORIA Lab. The ground is composed of a network of intelligent tiles. Each tile embeds a node that is able to store, compute and display information as well as communicate with neighboring tiles and nearby robots. LED lighting indicates that the tiles have detected the presence of people or robots (see `http://infositu.loria.fr`).

they visit cells. We use a swarm strategy to update the environment, while the D* algorithm computes the consequences of obstacle discovery immediately through its representation.

We plan to study how the propagation mechanisms of D* could be integrated into our approach to accelerate the APF convergence (e.g. by driving the agents for a while) and how D* could be distributed among a set of memory-less marking agents.

9 Towards a robotic implementation?

We now discuss how multi-agent algorithms using the marking of their environment could be deployed in real world with mobile robots. The question is how to read and write marks in the environment?

Several techniques for virtual marking have been proposed since the nineties. A typical approach is to use a shared representation of the environment, that agents can read and write through communication. The model proposed by Vaughan *et al.* (2000) is an interesting example of real-world implementation. This approach, however, changes the model to a centralized algorithm and requires robot localization. Payton *et al.* (2004) proposed another way to implement pheromone-based foraging, which requires only local communication between robots (i.e., no marking of the environment). In this approach, robots replace beacons as they can stop and form a network able to store pheromones and compute their diffusion (also called 'pheromone robots'). Hoff *et al.* (2010) proposed two foraging algorithms based on this principle. They also studied in simulation the problem of congestion, which degrades performance when robot density grows (as the approach requires numerous robots).

More recently, several studies investigated the possibility of writing chemical substances, even real pheromones, to mark the environment (Kowadlo and Russell, 2008; Purnamadjaja and Russell, 2010). It is however difficult to envisage applying such a solution to manage gradient construction due to substance volatility and to its environmental impact. In 2007, Garnier et al. (2007) proposed to display luminous trails in order to emulate pheromone deposit. The approach requires to set a video projector over the robots and to centralize the environment representation.

In these days of ubiquitous and cheap computing, the implementation of swarm and collective systems has become more realistic. A wireless network can be de-

ployed allowing robots to explore the environment using mark-based algorithms, see for instance (Mamei and Zambonelli, 2005) for an application of synthetic pheromone spreading. Another approach consists of installing in the floor a lattice of RFID or 'intelligent tiles' (Fig. 12) that allows robots to read and write information in the environment. See for instance (Johansson and Saffiotti, 2009; Herianto and Kurabayashi, 2009) and the iTiles model (Pepin et al., 2009).

A case study we plan to consider is the deployment of mobile robots in indoor environments where the ground is composed of intelligent tiles (Fig.12). Robots have to perform different tasks such as searching for objects and assisting people. In this context, robots do not know the location of obstacles such as furniture and objects. One challenge is to study how the environment can help robots in building routes, finding objects, etc.

10 Conclusions and future work

The study and the rewriting of the wavefront computation proposed by Barraquand *et al.* (1992) for path-planning has allowed us to define an efficient multi-agent foraging algorithm in finite grid environments.

We have shown that a set of simple agents can build the artificial potential field (APF) corresponding to the wavefront expansion only by reading and writing values in the environment. We proved that this distributed and asynchronous construction converges to the optimal APF in finite time.

We then defined a foraging algorithm exploiting this APF construction, that is, agents that search for and transport resources in unknown environments. The originality of our approach is to simultaneously build the wavefront APF while foraging, without depending on free parameters, contrary to pheromone-based approaches. This algorithm provides an efficient homing strategy, as agents simply need to follow the APF. By coloring return paths, agents are able to recover resource locations with a deterministic behavior. This defines a complete foraging algorithm that we called *c-marking agents*.

Simulation experiments showed the efficiency of the approach and in particular that cooperation can lead to superlinear performance. We compared our approach to a pheromone-based model, the ant algorithm. Results showed that our approach is more time efficient and scalable to the size of the environment. Another significant feature of our algorithm is its ability to operate in complex environments, such as a maze or an environment involving concave obstacles.

We now plan to study the robustness of our approach, and its ability to adapt to changes in dynamical environments, as ants do.

In order to improve the c-marking algorithm efficiency we think that the pseudo-random exploration can be enhanced by adapting some of the existing strategies used in covering tasks (Wagner et al. 2000; Glad et al. 2008). We are currently extending our approach to show that the proposed algorithm can be generalized to the construction of other potential fields and used with other problems or applications.

We will also test our approach using real robots interacting with an active floor, such as the iTiles. This is an important challenge as we seek new ways to bring bio-inspired algorithms to robots in real-world scenarios.

Acknowledgments

The authors wish to thank the editors and referees for their work in order to improve the analysis and the presentation of the results. We would like also to thank Olivier Buffet and Bruno Scherrer for their help in writing the paper, Anna Crowley and Julien Ponge for the English proofreading.

References

Arkin, R. (1998). *Behavior Based Robotics*. The MIT Press, Cambridge, MA.

Barraquand, J., Langlois, B., and Latombe, J. (1991). Numerical potential field techniques for robot path planning. In *'Robots in Unstructured Environments', 91 ICAR., Fith International Conference on Advanced robotics*, pages 1012–1017. IEEE Press, Picataway, NJ.

Barraquand, J., Langlois, B., and Latombe, J. (1992). Numerical potential field techniques for robot path planning. *IEEE Trans. on Systems, Man and Cybernetics*, 22(2):224–241.

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

Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J. M., and Duerinck, G. (1986). Random behaviour, amplification processes and number of participants : How they contribute to the foraging properties of ants. *Physica D: Nonlinear Phenomena*, 22(1-3):176–186.

Drogoul, A. and Ferber, J. (1992). From Tom Thumb to the dockers: Some experiments with foraging robots. In *2nd Int. Conf. On Simulation of Adaptative behavior*, pages 451–459. A Bradford Book / The MIT Press, Cambridge, MA.

Ferber, J. (1999). *Multi-Agent Systems, an introduction to Distributed Artificial Intelligence*. Addison-Wesley, London.

Garnier, S., Tache, F., Combe, M., Grimal, A., and Theraulaz, G. (2007). Alice in pheromone land: An experimental setup for the study of ant-like robots. In *Swarm Intelligence Symposium, IEEE SIS 2007*, pages 37–44. IEEE, Piscataway, NJ.

Glad, A., Simonin, O., Buffet, O., and Charpillet, F. (2008). Theoretical study of ant-based algorithms for multi-agent patrolling. In *Proceedings ECAI'08 18th European Conference on Artificial Intelligence*, pages 626–630. IOS Press, Amsterdam Netherlands.

Gutknecht, O. and Ferber, J. (2001). The MADKIT agent platform architecture. In *Infrastructure for Agents, LNAI 1887*, pages 48–55. Springer-Verlag Berlin Heidelberg.

Herianto and Kurabayashi, D. (2009). Realization of an artificial pheromone system in random data carriers using RFID tags for autonomous navigation. In *Proceedings ICRA'09 International Conference on Robotics and Automation*, pages 2288–2293. IEEE Press, Piscataway, NJ.

Hoff, N. R. I., Sagoff, A., Wood, R. J., and Nagpal., R. (2010). Two foraging algorithms for robot swarms using only local communication. In *Proceedings of IEEE-ROBIO 2010 : 2010 IEEE International Conference on Robotics and Biomimetics*, pages 123–130. IEEE Press, Piscataway, NJ.

Holldobler, B. and Wilson, E. (1990). *The Ants*. Harvard University Press, Cambridge, MA.

Johansson, R. and Saffiotti, A. (2009). Navigating by stigmergy: a realization on an RFID floor for minimalistic robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 245–252. IEEE Press, Picataway, NJ.

Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of IEEE Intern. Conf. on Robotics and Automation ICRA*, pages 500–505. The IEEE Computer Society Press, Los Alamitos, CA.

Koenig, S. and Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363.

Kowadlo, G. and Russell, R. A. (2008). Robot odor localization: A taxonomy and survey. *The International Journal of Robotic Research*, 27(8):869–894.

López-Ibañez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The iRace Package: Iterated Racing for Automatic Algorithm Configuration. Technical report, IRIDIA Technical Report Series No. 2011-004, Université Libre de Bruxelles.

Mamei, M. and Zambonelli, F. (2005). Spreading pheromones in everyday environments through RFID technology. In *In Proc. of the 2d IEEE Symposium on Swarm Intelligence*, pages 281–288. IEEE Computational Intelligence Society, New York.

Michel, F., Beurier, G., and Ferber, J. (2005). The turtlekit simulation platform: application to complex systems. In *Workshops Sessions, First International Conference on Signal-Image Technology and Internet-based Systems*, pages 122–127. IEEE, Los Alamitos, CA.

Monekosso, N., Remagnino, P., and Szarowicz, A. (2002). An improved Q-learning algorithm using synthetic pheromones. In *From theory to practice in multi-agent systems, second international workshop of central and eastern europe on multi-agent systems, CEEMAS 2001 Cracow. Revised Papers, Lecture Notes in Artificial Intelligence LNAI-2296*, pages 197–206. Springer-Verlag, Berlin, Germany.

Norris, J. R. (1998). *Markov Chains (Cambridge Series in Statistical and Probabilistic Mathematics No. 2)*. Cambridge University Press, UK.

Panait, L. and Luke, S. (2004). A pheromone-based utility model for collaborative foraging. In *Proc AAMAS'04 Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 36–43. ACM Press, New York.

Parunak, H. V. D. (1997). Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101.

Parunak, H. V. D., Purcell, M., and O'Connell, R. (2002). Digital pheromones for autonomous coordination of swarming UAV's. In *Proc. of AIAA First Technical Conference and Workshop on Unmanned Aerospace Vehicles, Systems, and Operations*. Published by American Institute of Aeronautics and Astronautics.

Pepin, N., Simonin, O., and Charpillet, F. (2009). Intelligent tiles: Putting situated multi-agents models in real world. In *Proceedings ICAART'09 International Conference on Agents and Artificial Intelligence*, pages 513–519. INSTICC Press, Setúbal, Portugal.

Purnamadjaja, A. H. and Russell, R. A. (2010). Bi-directional pheromone communication between robots. *Robotica*, 28(1):69–79.

Resnick, M. (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, Cambridge, MA.

Sauter, J. A., Matthews, R., Parunak, H. V. D., and Brueckner, S. (2002). Evolving adaptive pheromone path planning mechanisms. In *Proc. of AAMAS'02 First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 434–440. ACM Press, New York.

Sauter, J. A., Matthews, R., Parunak, H. V. D., and Brueckner, S. (2005). Performance of digital pheromones for swarming vehicle control. In *Proc. of AAMAS'05 Fouth International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 903–910. ACM Press, New York.

Sempe, F. and Drogoul, A. (2003). Adaptive patrol for a group of robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 2865–2869. IEEE Press, Piscataway, NJ.

Spears, W., Spears, D., Hamann, J., and Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2-3):137–162.

Steels, L. (1989). Cooperation between distributed agents through self-organization. In *Decentralized AI– Proc. of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 175–196. Elsevier Science Publishers, Amsterdam, The Netherlands.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1652–1659. Morgan Kaufmann Publishers, San Francisco, CA.

Vaughan, R. T., Støy, K., Sukhatme, G. S., and Matarić, M. J. (2000). Whistling in the dark: cooperative trail following in uncertain localization space. In *Proc. of the 4th International Conference on Autonomous Agents (Agents'00)*, pages 187–194. ACM Press, New York.

Wagner, I., Lindenbaum, M., and Bruckstein, A. M. (1998). Efficiently searching a graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence*, 24:211–223.

Wagner, I., Lindenbaum, M., and Bruckstein, A. M. (2000). MAC vs. PC: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of robotics Research*, 19(1):12–31.

Winfield, A. (2009). Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8, Proceedings of the 9th International Symposium on Distributed Autonomous Robotic Systems (DARS 2008)*, pages 185–192. Springer, Berlin Heidelberg, Germany.

Zhu, Q., Yan, Y., and Xing, Z. (2006). Robot path planning based on artificial potential field approach with simulated annealing. In *Proceedings ISDA'06 Sixth International Conference on Intelligent Systems Design and Applications, vol. 2*, pages 622–627. IEEE Computer Society Press, Los Alamitos, CA.