An Energy-Aware Algorithm for Large Scale Foraging Systems

Ouarda Zedadra, Hamid Seridi, Nicolas Jouandeau, Giancarlo Fortino

DOI 10.12694/scpe.v16i4.1133
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 4, pp. 449–466. http://www.scpe.org

AN ENERGY-AWARE ALGORITHM FOR LARGE SCALE FORAGING SYSTEMS

OUARDA ZEDADRA[*], HAMID SERIDI[†], NICOLAS JOUANDEAU[‡], AND GIANCARLO FORTINO[§]

Abstract. The foraging task is one of the canonical testbeds for cooperative robotics, in which a collection of coordinated robots have to find and transport one or more objects to one or more specific storage points. Swarm robotics has been widely considered in such situations, due to its strengths such as robustness, simplicity and scalability. Typical multi-robot foraging systems currently consider tens to hundreds of agents. This paper presents a new algorithm called Energy-aware Cooperative Switching Algorithm for Foraging (EC-SAF) that manages thousands of robots. We investigate therefore the scalability of EC-SAF algorithm and the parameters that can affect energy efficiency overtime. Results indicate that EC-SAF is scalable and effective in reducing swarm energy consumption compared to an energy-aware version of the reference well-known c-marking algorithm (Ec-marking).

Key words: Swarm intelligence, swarm robotics, multi-agent foraging, energy awareness, energy efficiency.

AMS subject classifications. 68W05

1. Introduction.

Swarm Intelligence comes from biological insights related to the capabilities that social insects possess to solve daily-life problems within their colonies [1]. Social insects (ants, bees, termites) provide complex collective behavior to accomplish complex tasks that can exceed individual capacities [2]. They usually lack sophisticated communication capabilities as well as sensor information [3], in opposite to intelligent and proactive individuals which are provided with sophisticated communication tools and sensors [7]. Inspirations from these biological systems known as pheromone-based coordination mechanisms have been presented for patrolling [8], localization [9] and mapping and exploration [10]. Swarm robotic is interested in the implementation of systems which are composed of thousands simple robots rather than one single complex robot [11] [12]. The simplicity of these individuals presents a highly structured social organization which can accomplish complex tasks that in far exceed the individual capacities of a single individual [13] [14]. The collection carries out complex tasks based on simple rules, without spending much computational power and much energy [15]. Behavior-based algorithms have been developed to manage homogeneously swarms of individuals and to achieve more complex tasks that can involve dynamic hierarchies [16] [17].

Foraging is a benchmark problem widely studied by swarm robotics: it is a complex task involving the coordination of several sub-tasks including efficient exploration, physical collection, transport, homing and depositing [18]. Foraging can be achieved by one single robot or by a collection of cooperative robots. By contrast, some animals are foraging individually for food [19] [20]; while others (as social insects) are foraging and recruiting collaborators by sharing information via stigmergy [21] [22] or direct signalling [23] [3]. In a foraging task, robots repeatedly collect objects from unknown locations and drop them of at another location. To work for long periods, they must have a means of obtaining more energy when their stored energy is exhausted. Time spent in searching, homing, recharging and returning to work can contain useless additional time and a current challenge in multi-robot foraging is to minimize it.

We investigate in this paper the energy problem within multi-agent foraging task through the analysis of our energy-aware algorithm (EC-SAF algorithm [4]). We tested it in large-scale foraging systems with hundreds and thousands of agents where we consider a wider range of system sizes than the ones considered in literature works and in our previous work [4]. In comparison to the literature works, we contributed at:(1) reducing the time spent in search by using our search algorithm (Stigmergic Multi-Ant Search Area algorithm–S-MASA [5]) which provides large dispersion, thus quick search and shortest paths relaying food to nest location for homing, (2) reducing the time needed for transporting food by allowing recruitment to found food defined in our previous foraging algorithm (*C-SAF algorithm* [6]).

The remainder of the paper is organized as follows: background works are presented in Section 2. Then

[*]LabSTIC, 8 may 1945 University, P.O.Box 401, 24000 Guelma. Department of computer science Badji Mokhtar, Annaba University, P.O.Box 12, 23000 Annaba, Algeria(zedadra_nawel1@yahoo.fr).
[†]LabSTIC, 8 may 1945 University, P.O.Box 401, 24000 Guelma, Algeria
[‡]LIASD, Paris 8 University, Saint Denis 93526, France
[§]DIMES, Universita' della Calabria, Via P. Bucci, cubo 41c - 87036 - Rende (CS) - Italy

EC-SAF and Ec-marking algorithms are presented in Section 3. In Section 4, we define the performance indices, describe simulation scenarios and compare the results of the two algorithms. Finally, Section 5 concludes the paper and shows some future works.

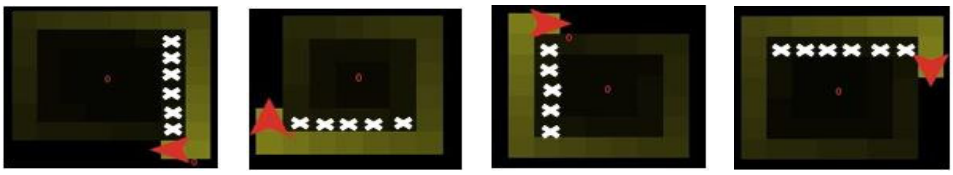
2. Background. The most common strategies for powering long-lived autonomous robots are capturing ambient energy and managing energy from recharging stations.

Capturing ambient energy directly from the environment, also known as energy scavenging [24]. A robot have to choose between searching and resting to conserve some energy and it accumulates some units of energy for each retrieved food. Krieger and Billeter [25] adopt a threshold-based approach to allocate foraging or resting tasks. They distinguish nest and robots energies. If nest energy is under a predefined threshold, then robots search for food. If one robot energy is under a predefined threshold, then it stays in the nest to recharge. Labella et al. [26] propose an adaptive approach to change the ratio of foragers to resters by adjusting the probability of leaving home. Liu et al. [27] consider collisions and object retrieval. The successes are accounted for individuals and teams in [26]. Efficient foraging is a special case, in which several types of objects, each provides a specific amount of energy are scattered in the environment and robots should choose between transporting a found food or continue searching for another that provides more energy. In [28] authors design a mathematical model of multi-foraging that predicts with a good confidence the optimal behavior of the robots that leads to maximize the energy of the group.

Transferring energy from a recharging station. Several options are possible in this case: (i) Working robots perform their work until their energy falls below a given threshold. At this time, they return to recharging station. Such decision can also be achieved with a behavior model [3]. While in [29], authors propose a distributed behavioral model for foraging regarding role division and search space division for improving energy efficiency; (ii) Working robots can stay at the working site permanently, while special dock robots visit them periodically to provide them with energy. In [30], a dock station is an autonomous robot with unlimited energy store, that searches gradually to improve its position regarding the position of working robots. In [31], the on-board recharging electronics and intelligent docking stations enable the robots to perform autonomous recharging; and (iii) Robots could transfer the energy also between them by comparing their energy's level. Robots could transfer the energy between each other in a trophallaxis-like manner, where robots with higher energy status can share energy with others having lower energy status [32].

In the following subsections, we present the background works for the energy-aware algorithms proposed in this paper: *S-MASA algorithm* [5], *C-SAF algorithm* [6] and *c-marking algorithm* [33] (Section 2.1, Section 2.2 and Section 2.3 respectively).

2.1. S-MASA Algorithm. The S-MASA algorithm is an exploration strategy inspired by the behavior of ants and water vortex dynamics. It was first applied to multi-target search and coverage tasks [5], and to multi-agent foraging tasks in [34] [6]. Using S-MASA algorithm as a search strategy has contributed to: (1) speed up the exploration by avoiding the already visited cells (cells containing pheromone), and (2) speed up the homing process, because optimal paths are simultaneously built while agents explore their environment (a wavefront of pheromone concentration is created). According to the coordination rules of S-MASA algorithm in Fig. 2.1, the agent changes its heading if there is no pheromone in its right cell (cell not yet visited), else it keeps going in the current heading.



(a) (b) (c) (d)

FIG. 2.1. *S-MASA coordination principle represents the changing headings: (a) from 180 to 270 (b) from 270 to 0 (c) from 0 to 90 (d) from 90 to 180, where white crosses represent already visited cells*

2.2. C-SAF Algorithm. C-SAF algorithm [6] is a multi-agent foraging algorithm that uses S-MASA as search strategy. Agents in C-SAF algorithm use a three layered subsumption architecture [35] where each layer implements a particular behavior: *Environment exploration* is the lowest priority layer in this architecture. It consists in exploring the environment, therefore, it includes the states *Choose-Next-Patch* and *Look-for-Food*. *Food exploitation* consists in exploiting food when it is found, it envelops the states *Pick-Food*, *Return-to-Nest*, *Return-and-Color*, *At-Home*, *Climb* and *Remove-Trail*. *Obstacle avoidance* is the higher priority layer, it implements the obstacle avoidance behavior. Higher priority layers are able to subsume lower levels in order to create viable behavior (see Fig. 2.2 for an illustration of the architecture). The behavioral model of the C-SAF foraging agents is depicted by Fig. 2.3, without the bold guards and the dashed transitions.

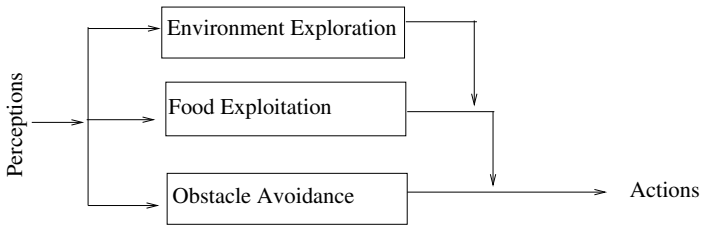


FIG. 2.2. *Subsumption architecture of C-SAF foraging agents*

2.3. The c-marking Foraging Algorithm. The c-marking foraging algorithm is a distributed parameter-free algorithm for multi-agent foraging [33]. It is a distributed and asynchronous version of wavefront algorithm [36]. It allows agents to build non optimal paths simultaneously while exploring by computing a wavefront expansion from the base station (sink). Agents use this wavefront to return to the sink. They move at each step to the cell with the smallest Artificial Potential Field (APF) value. Pseudo random walk is used by agents as a search strategy. Agents try to move to a non neighboring marked cell. If all neighboring cells are marked, then they move randomly. Unfortunately, the convergence of APF to its optimal values takes huge time. Agents therefore need to visit the same cell several times. The c-marking foraging algorithm have multiple advantages: (1) agents build simultaneously paths when they explore and (2) it is very robust to agents' failure and to complex environments. However, it provides some drawbacks: (1) paths are of long length specially when the number of agents is not sufficient to sustain the cells corresponding to the wavefront, this will increase dramatically the foraging time, (2) revisiting already visited cells in static environments costs in number of steps. The behavioral model of the c-marking foraging agents is depicted by Fig. 2.4, without the filled state, bold guards and the transitions (dashed arrows).

3. An Energy-aware Multi-Agent Foraging Algorithm. In this Section, we present the EC-SAF (Energy-aware Cooperative Switching Algorithm for Foraging) algorithm. Specifically, we model the behavior of foraging agents as a finite state machine. We also present the Ec-marking algorithm, which is an energy-aware version of the well-known c-marking algorithm [33]. EC-SAF and Ec-marking will be compared through simulations in Section 4.

3.1. Modeling of the Components. The components of the two multi-agent systems (environment, agent and pheromone) were modeled as in below:

- *Environment Model:* The search space is an N x N grid world with several food locations, one or multiple sinks (nest), obstacles and pheromone markings. It is divided into equal squares in a Cartesian coordinate system. Grid maps are thought to an efficient metric for navigation in large-scale. In robotics, it is often used for solving tasks like path planning [37], localization [38] and search and surveillance in dynamic environments [39]. It is efficient that the full grid is square and all cells are the same size of an agent. N food with M items (we refer to it as concentration) are located randomly on grids. Obstacles with rectangular or square shapes take place on some fixed cells, the nest is at the center (in a multi-sink space, sinks take specific positions to guarantee the completeness of the algorithms).
- *Agent Model:* All agents of the colony are homogeneous with limited sensing and actuation capabilities. They start all from a predefined position (the nest) and headings (0, 90, 180 and 270). An agent can

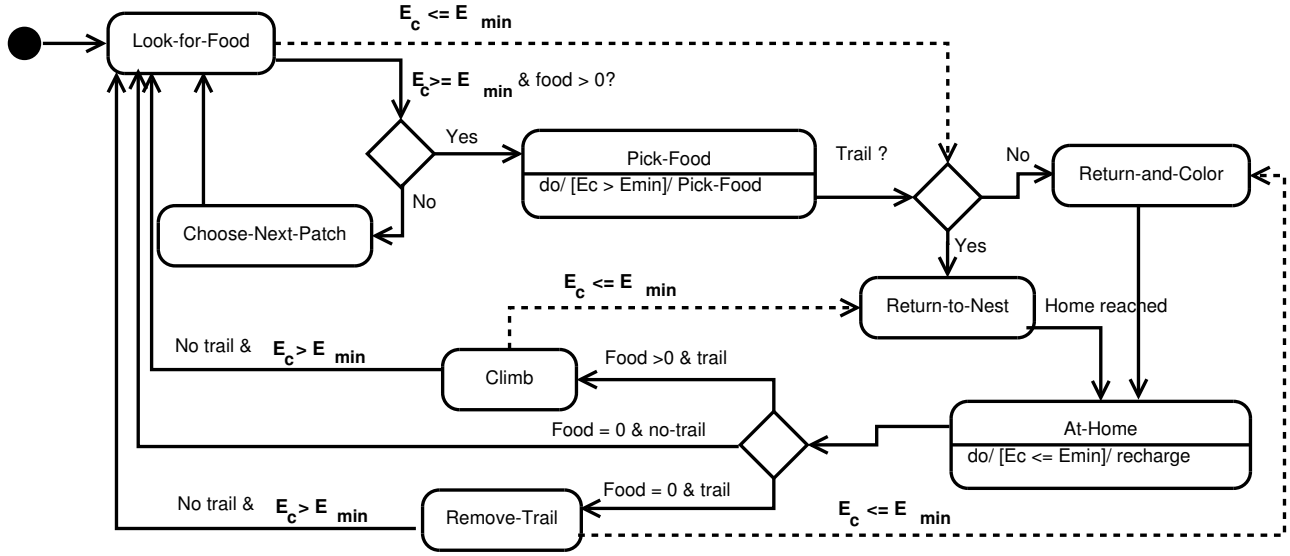


FIG. 2.3. *the state machine of a foraging agent. Dashed arrows (transitions), bold guards and the internal actions in the states represent the energy-aware behavior of agents*

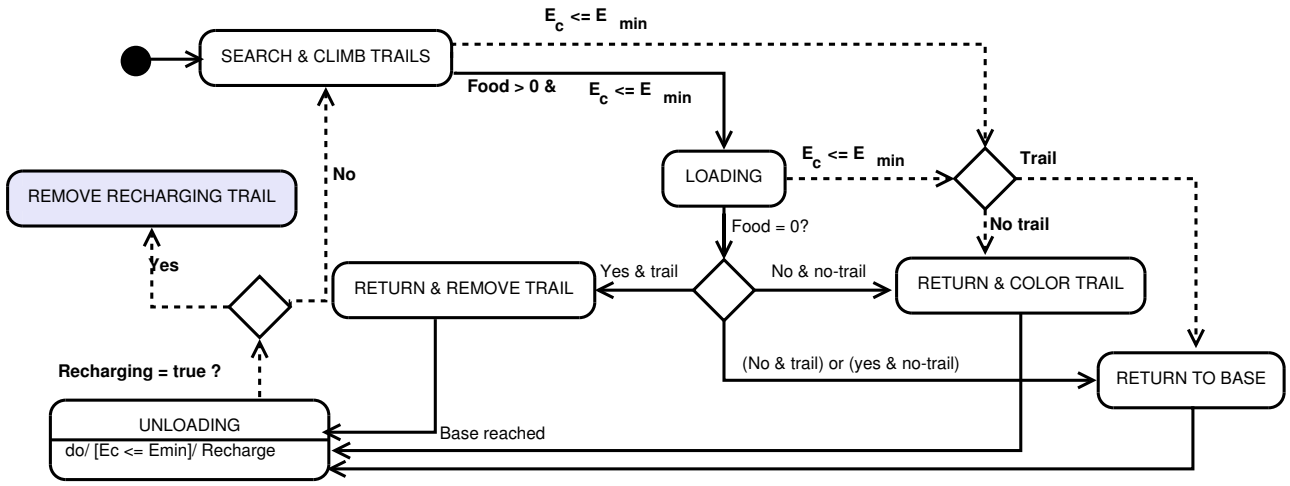


FIG. 2.4. *State transition diagram of an Ec-marking agent, where colored state, dashed arrows (transitions) and bold guards are related to energy-aware behavior of agents*

carry out the following sequence of actions:

- Senses whether a food item exists on the current cell and if there exists a pheromone in the four neighboring cells, whether it is at the nest, and it recognized whether it is carrying food or not.
- Deposits a limited amount of pheromone in the current cell.
- Picks up (loads) or puts down (unloads) a food item, when it is at food and carries no food, and when it is at nest and carrying food respectively.
- Selects and carries out actions according to the surrounding events. The agent have four different behaviors corresponding to the four models (C-SAF, c-marking, EC-SAF and Ec-marking) and the set of actions to execute differ from a model to another.

• *Pheromone Releasing and Evaporation:* The agent is able to secrete several types of pheromone, which

did not affect each other, at the same cell in the environment. Each forager is capable of depositing three types of pheromone. The first type, is dynamic and subsequently gradually evaporates (according to Eq. 3.1), used to mark already visited cells. The two others, are static with no diffusion and evaporation properties, one is used to keep track of the food location (food trails in yellow color) and the other to attract other agents to food locations (recruitment trails in brown color).

$$\Gamma_i(t+1) = \gamma_i(t) - p * \gamma_i(t) \tag{3.1}$$

where p is a coefficient which represents the evaporation rate of trail between time t and *(t+1)*.

3.2. EC-SAF Algorithm. The EC-SAF algorithm extends the C-SAF algorithm [6] to consider energy limitation in a foraging system, where agents use the existing set of states (equivalent to return and color / remove trails) with some additional transitions and guards. We added a layer in the previous subsumption architecture (Fig. 2.2), called *Recharging energy* layer which envelops the set of states that allow agents to return home to recharge when their energy falls below a given threshold, it includes the states *Return-to-Nest*, *Return-and-Color*, *Remove-Trail* and *At-Home* (see Fig. 3.1 for the subsumption architecture with the new layer). The behavior of our energy-aware foraging agents is shown by the state machine in Fig. 2.3 where dashed arrows (transitions), bold guards and actions in the states are used when the current energy of an agent (E_c) falls below the fixed threshold (E_{min}). States are described below and the EC-SAF Algorithm is given by Algorithm 1.

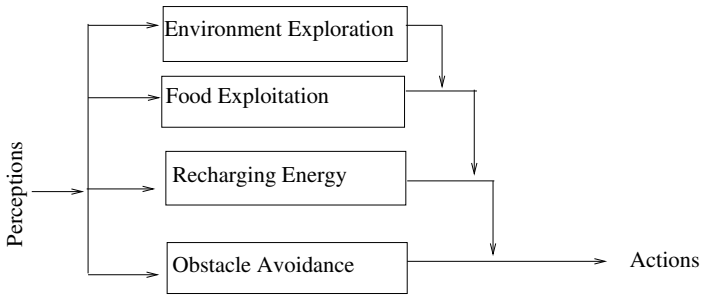


FIG. 3.1. *Subsumption architecture of EC-SAF foraging agents*

Look-for-Food: If $E_c > E_{min}$ and there exists a food here, agent executes *Pick-Food* state, while if there exists no food it executes *Choose-Next-Patch* state. If there exists a trail and its $E_c <= E_{min}$, it turns to *Return-to-Nest* state, or to *Return-and-Color* state, if there exists no trail.

Choose-Next-Patch: If an obstacle is detected, the agent calls the procedure Avoid_Obstacle(). If no obstacle is there, the agent climbs the brown trail to reach the food location if there exists one, it spreads then the information to its left cell. It lays a limited amount of pheromone in current cell, adjusts its heading by executing S-MASA Algorithm [5] and moves one step forward. It turns automatically when finished to *Look-for-Food* state.

Pick-Food: If $E_c > E_{min}$, agent picks a given amount of food and spreads the information to its left cell. However, if $E_c <= E_{min}$ it does not pick food. It executes in the two cases *Return-to-Nest* state, if there exists a trail or *Return-and-Color* state if there exists no trail.

Return-to-Nest: The agent moves to one of colored neighboring cells with the lowest pheromone value. It remains in this state until home is reached, it turns then to *At-Home* state. ;

Return-and-Color: The agent moves to one of the four neighboring cells with the lowest pheromone value and marks its trail with yellow and remains in this state until it reaches the home; it turns then to the *At-Home* state.

At-Home: The agent unloads food if it carries one. If its current energy *(E_c)* is below *(E_{min})*, it recharges its energy to the maximum amount E_{max}. It goes to *Climb* state if there exists a trail and the amount of food is > 0. If amount of food is $= 0$ and there is a trail, it executes *Remove-Trail* state, else it turns to *Look-for-Food* state.

Look-for-Food
if *(E_c <= E_{min})* **then**
  if *(∃ trail)* **then goto** *Return-to-Nest*;
  else goto *Return-and-Color*;
else
  if *(food > 0)* **then** Diffuse(P); **goto** *Pick-Food*;
  else goto *Choose-Next-Patch*;

Choose-Next-Patch
if *(obstacle detected)* **then** Avoid_Obstacle();
else
  if *(brown P here)* **and** *(brown P in right cell)* **then**
    Diffuse(P); move to food location using brown cells;
  else
    if *(brown P here) and (no brown P in right cell)* **then**
      remove brown trail;
    else
      Lay(P);Detect_And_Adjust_Heading() Update(P); Move();
goto *Look-for-Food*;

Pick-Food
if *(E_c > E_{min})* **then** pick up a given amount of food; Diffuse(P);
if *(∃ trail)* **then goto** *Return-to-Nest*;
else goto *Return-and-Color*;

Return-to-Nest
while *home not reached* **do**
  move to a colored neighboring cell with the lowest P;
goto *AT-HOME*;

Return-and-Color
while *home not reached* **do**
  move to a neighboring cell with the lowest P Color that cell to a specific trail color (yellow);
goto *At-Home*

At-Home
unload food **if** *(E_c <= E_{min})* **then** recharge E_c to E_{max};
if *(∃ trail* **and** *food > 0)* **then goto** *Climb*;
else if *(∃ trail* **and** *food = 0)* **then goto** *Remove-Trail*;
else goto *Look-for-Food*;

Climb
while ∃ *trail* **do**
  if *(E_c <= E_{min})* **then goto** *Return-to-Nest*;
  else move to neighboring colored cell with the greater value of P;
goto *Look-for-Food*;

Remove-Trail
while ∃ *trail* **do**
  if *(E_c <= E_{min})* **then goto** *Return-and-Color*;
  else move to neighboring colored cell with the greater value of P and update its color to the default one (black);
goto *Look-for-Food*;

Algorithm 1: EC-SAF Algorithm, where pheromone is noted P

Climb: Agent moves to one of its four colored neighbors with a pheromone value greater than the pheromone value of the current cell. It remains in this state until no colored cell (yellow trail) exists and its $E_c > E_{min}$, it turns then to the *Look-for-Food* state. If its $E_c <= E_{min}$ and since there exists a trail, it executes the *Return-to-Nest* state in order to return home for recharging its energy.

Remove-Trail: The agent moves to a colored cell with the greatest pheromone value and resets its color to the default color (black). It remains in this state until no colored cell is found and $E_c > E_{min}$, it turns then to the *Look-for-Food* state. If $E_c <= E_{min}$, the agent returns to home to recharge and executes the *Return-and-Color* state since it already removed the existing trail and to keep track of the last position from where it will continue removing after it recharges its energy.

3.3. Ec-marking Algorithm. The Ec-marking algorithm extends the c-marking algorithm [33] to deal with energy limitation. Ec-marking agents behavior is given by Fig. 2.4 (the extensions made are represented by filled states, dashed transitions and bold guards), while using the rules in Algorithm 2. Agents while exploring the environment build simultaneously paths between food and nest which results in building an ascending Artificial Potential Field (APF) incrementally [33]. Agents use pseudo random walk as search strategy, which results in non-optimal paths and makes the convergence of the APF to its optimal value very slow. We use the same set of states as in c-marking to return home for recharging (states corresponding to return and color or return to base), while for removing the energy trails, we define a new state called *REMOVE RECHARGING TRAIL*. This last state is activated when agents finish recharging their energy from home, they climb then the trail and remove until there is no trail, they resume then their search process.

4. Performance Evaluation. In this Section, we discuss the performance of the four algorithms (EC-SAF [Algorithm 1], C-SAF [6], c-marking [33] and Ec-marking [Algorithm 2]) in obstacle-free and obstacle environments. We present in Section 4.1 the performance indices and simulation parameters used to evaluate the algorithms. On the basis of analysis of an execution example for the four algorithms in Section 4.2, we compared only E-C-SAF and Ec-marking algorithms in Section 4.4, we describe the scenarios used for simulations in Section 4.3.

In each simulation several parameters are to be set:

- Agent parameters: the number of agents that participate at each simulation (*agent number*) and the amount of food (in units) that an agent can transport at each time (*agent capacity*).
- World parameters: include the dimension of the search space which is a grid of $N \times N$ *cells* (*world size*). *World complexity* which can be obstacle-free or obstacle environment. *Sink number* which is the number of the home or base stations to where agents return food and recharge energy.
- Food parameters: the number of food locations (*food density*) which are located at fixed positions. Each food location contains a limited amount of food called *food concentration*.
- E_{max}: is the maximum value that an agent can recharge when it reaches home and its E_c is below the threshold E_{min}. An analysis of the effect of different E_{max} on overall search time and the total energy consumed in EC-SAF algorithm is given in Table 4.1. When E_{max} is set to smaller value (1000 and 3000 units) agents return home several times to recharge (18 to 5 times), increasing the search time and the energy consumed, thus the search becomes inefficient. It is set to 5000 energy units, since number of returns to recharge is small (2 times) thus doesn't consume much energy and search time.
- E_c: represents the current energy that an agent have at time *(t)*. It is set initially to E_{max} that is 5000 energy units. When $E_c <= E_{min}$, agents need to return home to recharge energy.
- E_{min}: is the threshold that activates the return to home and recharge energy behaviors. It is fixed experimentally to a value that allows agents to get home before their death ($E_c = 0$) (Table 4.2). With 100 units, agents can't reach home because the path is longer and needs more energy units. When we increased E_{min} to 200 units, agents could reach the home the first return, but in the second return they couldn't reach it. We increased then energy to 300 units and it was sufficient to return home for 2 times to recharge (and food is reached in this stage). The pseudo random walk in Ec-marking makes the measuring of the E_{min} very difficult, sometimes food is rapidly located and paths are close to optimal, thus the 300 energy units are sufficient to return home to recharge, but in other cases, paths are not optimal and agents need more than 300 energy units to reach home. To deal with such cases, we fixed the E_{min} to 500 units.

SEARCH & CLIMB TRAIL(Repeat)
if *(E_c <= E_{min})* **then**
  if *(∃ trail)* **then goto** *RETURN TO BASE*;
  else goto *RETURN & COLOR TRAIL*;
else
  if *(food exist in neighboring cell)* **then**
    move into that cell; **goto** *LOADING*;
  else
    if *(∃ trail)* **then**
      move to the highest valued colored neighboring cell;
    else goto *EXPLORATION & APF CONSTRUCTION*;

LOADING
if *(E_c <= E_{min})* **then**
  if *(∃ trail)* **then goto** *RETURN TO BASE*;
  else goto *RETURN & COLOR TRAIL*;
else
  Pick up a quantity Q_{max} of food;
  if *(food is not exhausted)* **then**
    if *(∃ trail)* **then goto** *RETURN TO BASE*;
    else goto *RETURN & COLOR TRAIL*;
  else goto *RETURN-AND-REMOVE-TRAIL*;

RETURN & COLOR TRAIL
color the cell in a specific color and update value;
if *(base reached)* **then goto** *UNLOADING*;
else move to a neighboring cell with the *smallest APF value*;

RETURN & REMOVE TRAIL
if *(cell has the trail color)* **then** remove this color and *UPDATE VALUE*;
if *(base is reached)* **then goto** *UNLOADING*;
if *(there exists a neighboring cell with trail color)* **then**
  move to neighboring trail cell;
else move to the smallest valued neighboring cell;

UNLOADING
unload food;
if *(E_c <= E_{min})* **then** recharge a limited amount of energy;
if *(recharging = true)* **then goto** *REMOVE RECHARGING TRAIL*;
else goto *SEARCH & CLIMB TRAIL*;

REMOVE RECHARGING TRAIL
while ∃ *trail* **do**
  if *(E_c <= E_{min})* **then goto** *RETURN & COLOR*;
  else move to neighboring colored cell with the highest APF value;
goto *SEARCH & CLIMB TRAIL*;

Algorithm 2: Ec-marking Algorithm

4.1. Performance Indices and Simulation Parameters. We analyze in this Section the performances of the two algorithms in large-scale systems within the foraging task. We used therefore, large number of agents and large environment sizes with the aim to: (1) test and conclude the benefit of providing agents with energy-

TABLE 4.1
E_{max} Analysis

	1000	3000	5000	7000	9000
Search time	7761,4	5771,3	2530,1	2231,4	2030,1
Energy Consumed	12475	10550	10125	9935	6095
Number of returns	18	5	2	1	0

TABLE 4.2
E_{min} Analysis

	100	200	300	400	500
Number of returns	0	1	2	2	2
Success to get home?	No	No	Yes	Yes	Yes

aware behaviors to sense and manage their energy, thus go to an energy-aware version of the C-SAF algorithm, (2) test the efficiency of the proposed algorithm through the total amount of energy consumption regarding the amount of food foraged, and (3) observe the evolution of energy consumed by the swarm overtime. We used the performance indices *Total Food Returned*, *Energy Efficiency* [29], *Energy Efficiency overtime*, *Total Energy Consumed* and *Finish foraging time*.

- **Total Food Returned**: is the total amount of food (in units) returned over a given elapsed time.
- **Total Energy Consumed**: is the total energy spent by all agents to search and exhaust all the food locations.
- **Energy Efficiency**: is the average energy consumed for foraging food. It is calculated according to Eq. 4.1. Since the *Total Energy Consumed* increases with the time, if $E_{eff1} > E_{eff2}$, it means that E_{eff2} is better.

$$E_{eff} = \frac{TotalEnergyConsumed}{TotalFoodReturned} \tag{4.1}$$

- **Energy Efficiency Overtime**: is the average energy consumed until time t to forage an amount of food units until time t. It is calculated according to Eq. 4.2.

$$E_{eff}(t) = \frac{TotalEnergyConsumed(t)}{TotalFoodReturned(t)} \tag{4.2}$$

  where: *TotalEnergyConsumed(t)* is the total energy consumed until a given elapsed time t [0..t] and *TotalFoodReturned(t)* is the number of food Foraged until a given elapsed time t [0..t].
- **Finish foraging time** (T_{forag}): The amount of time in seconds needed to finish the foraging mission. It is when all the food sites are discovered and exhausted.

The energy consumption of an agent at each state is defined as estimation of the power of real robot equipment (such as motor, sensor and processor) required to achieve that state. It is inspired by the B-swarm model [3]. The energy consumption settings of a robot are described in Table 4.3 for the two energy-aware algorithms.

4.2. Example of Simulation Execution. C-SAF agents have limited energy but no recharging ability, so after their energy stores are empty they die while EC-SAF agents have limited energy and recharging behaviors, that allow them to work for long time. Simulation is based on Netlogo [40]. We used a preliminary scenario to test the benefit of providing agents with energy-aware behaviors when they have limited energy stores. We used environments with a 1000x1000 cells with: one sink at the center, one food location that contains 500 units of food, to exclude the impact of food position, the food is placed at fixed position for all simulations; 300 agents with a capacity of transport of one unit at each time. We calculated the total amount of food returned

TABLE 4.3
The agent energy consumption of each state

States	Energy consumed (unit/simulation update)
EC-SAF States	
At-Home, Choose-Next-Patch, Pick-Food,	5
Remove-Trail, Return-and-Color	0
Climb, Return-to-Nest	1
Avoid-Obstacle	3
Ec-marking States	
SEARCH, LOADING, UNLOADING,	5
RETURN & REMOVE TRAIL	0
RETURN & COLOR TRAIL	0
REMOVE RECHARGING TRAIL	0
CLIMB TRAIL, RETURN TO BASE	1
Avoid-Obstacle	3

until the completion foraging time for the algorithm that takes the longest time (Ec-markings algorithms). The obtained results with the four algorithms (C-SAF, EC-SAF, c-marking and Ec-marking) are depicted by Fig. 4.1, where we limited the energy stores in C-SAF and c-marking to 2000 units and the E_{min} to 500 units. From Fig. 4.1, we observe that the total food returned increases overtime in the four algorithms when agents have enough energy. However, when the agent energy is over in algorithms C-SAF and c-marking, agents die and there will be no changes in *Total Food Returned*. While in EC-SAF and Ec-marking, they return home after their energy fall below E_{min}, recharge and resume their tasks. EC-SAF reaches the total amount of food at 1600 seconds (2100 seconds in obstacle environment), thus it stops search, while Ec-marking reaches it at 2300 seconds (3300 seconds in obstacle environment). Agents spend more time and consume more energy, specially in c-marking algorithms since paths are not optimal, while in EC-SAF agents use optimal paths, which contributes at reducing the recharging the overall and the energy consumed.

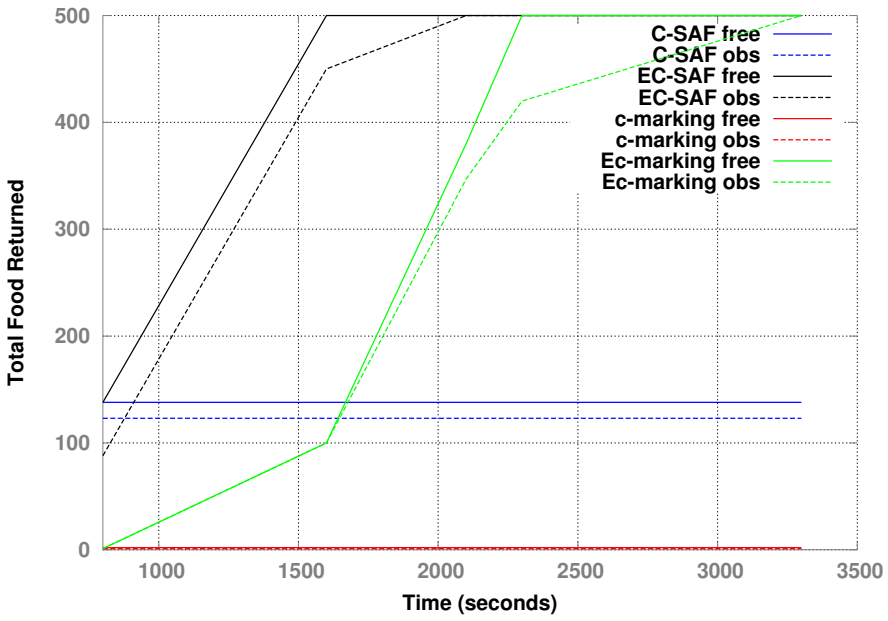
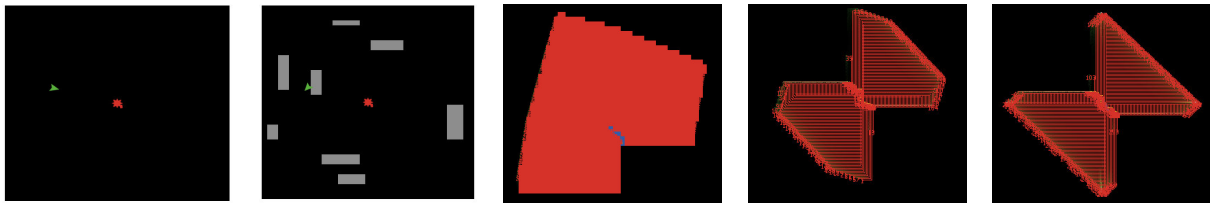


FIG. 4.1. *Total Food Returned by EC-SAF, C-SAF, Ec-marking and c-marking agents until the finish foraging time in obstacle-free and obstacle environments*



(a) (b) (c) (d) (e)

FIG. 4.2. *World setups used in simulations. (a) obstacle-free environment, (b) obstacle environment, where red arrows are agents, green arrows are food locations, gray blocks are obstacles and pink squares are sinks. (c)(d)(e) present samples at different stages where laden agents are in the middle line, and searcher agents spread themselves diagonally on sides.*

TABLE 4.4
Parameters of scenario 1, scenario 2 and scenario 3

Parameter		**Value**
Scenario 1	E_{eff} Analysis	
World size		100x100 cells – 1000x1000 cells
Number of agents		100 – 1000 agent
Food density		1 – 10 locations
Food concentration		500 units
Agent capacity		1 unit
Sinks number		1 – 64 sink
Scenario 2	*Total Energy Consumed Analysis*	
World size		1000x1000 cells
Number of agents		300 agent
Food density		2 locations
Food concentration		500 units
Agent capacity		1 unit
Sinks number		1 sink
Scenario 3	*Scalability Analysis*	
Agent Density		0.00001– 0.01
Food density		2 location
Food concentration		400 units
Agent capacity		1–10 unit
Sinks number		1 sink

Figure 4.2 shows some of the world setups used in the three scenarios discussed above. An obstacle-free and obstacle environments with one food location and one sink at the center of world in Figs 4.2(a) and 4.2(b) respectively. While we present samples of simulation execution in obstacle-free environment with one sink and one location in Fig. 4.2(c) and with two food locations in Figs 4.2(d) and 4.2(e) where agents in the middle are transporting food and agents at the sides still searching.

4.3. Simulation Scenarios. Several kinds of simulations are carried out in this paper. The parameters which we tested and analyzed are reported in Table 4.4. We proposed three fundamental scenarios:

- Scenario 1 is to test which of the parameters presented in Section 4 can affect the *Energy Efficiency*. It envelops 5 sub-scenarios. Sub-scenario 1 is to show $E_{eff}(t)$ pattern overtime. Sub-scenario 2 is to analyze the impact of *agents number* on performance, which number gives less E_{eff}. Sub-scenario 3 is to test how *world size* can affect the E_{eff} and is there any minimal size under which a number of agents can carry efficiently the foraging task. In order to analyze whether clustering the food units in

one location or distributing it over several locations improves performances, we defined sub-scenario 4 where we varied *food density* from 1 to 10. To test the effect of dividing the environment into smaller search spaces, we defined sub-scenario 5, in which we varied number of sinks from 1 (one search space) to 64.

- Scenario 2 is defined to show the *Total Energy Consumed* by the swarm of agents over the foraging task.
- Scenario 3 is defined to study the scalability of our algorithm and whether varying the *agent density* affects positively or negatively the E_{eff}. We used three sub-scenarios. Where we define in each one *Agent Density* as *Number of Agents / World Size*. In the first sub-scenario, we kept the *agent density* fixed to a given value while varying *agents number* and *world size* simultaneously. In the second, we increase the *agent density* by keeping *agents number* fixed and increasing *world size*. In the third, we decrease the *agent density* by increasing the *agents number* and keeping *world size* fixed.

4.4. Results Analysis.

4.4.1. Results in Scenario 1. Results in Fig. 4.3 show a decrease in $E_{eff}(t)$ in the two algorithms. In EC-SAF, when agents search and depose pheromone, they consume more energy, but when food is located the amount of food returned increases, thus the ratio decreases. However, in Ec-marking, the $E_{eff}(t)$ is high because of the pseudo random walk which slows the search process, after the food is located, $E_{eff}(t)$ decreases dramatically overtime since the amount of food returned increases. Results obtained with EC-SAF (also with Ec-marking) in obstacle-free and obstacle environment are very close and their curves overlap in Fig. 4.4.

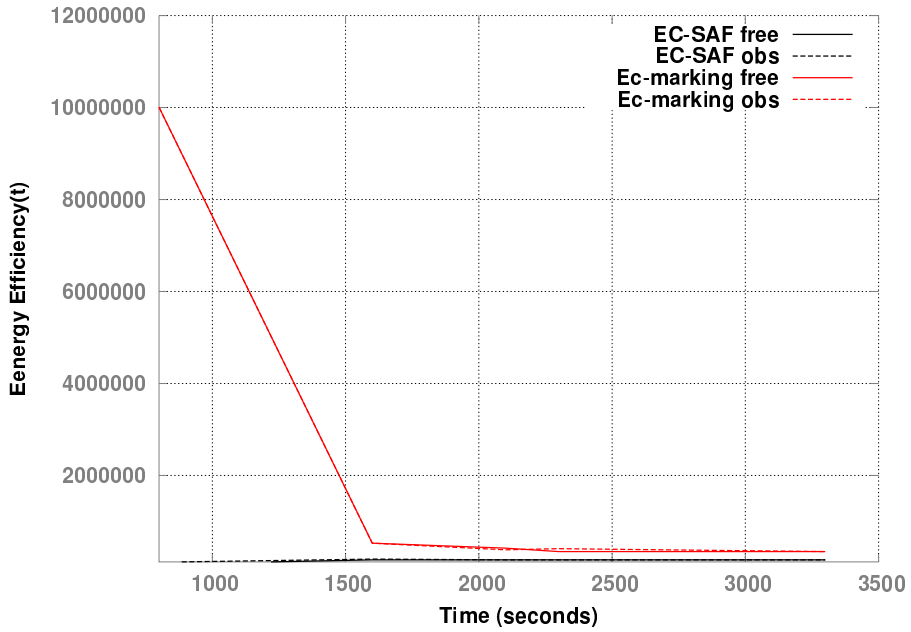


FIG. 4.3. *Variation of* $E_{eff}(t)$ *overtime*

Figure 4.4(a) shows the results obtained by EC-SAF and Ec-marking algorithms when varying agent number. The growth of E_{eff} in EC-SAF is constant and slow. While, in Ec-marking E_{eff} is inconstant. It increases and decreases overtime, higher with 100 agents, where they spend more time in search (finish foraging time is 9907 seconds), thus consume more energy. It is much less with 300 agents (finish foraging time 2325 seconds only). The quick search provided by S-MASA algorithm [5] in EC-SAF contributes at reducing search time, thus energy consumption and E_{eff}.

E_{eff} in EC-SAF is reduced with each increase in world size. This last affects the results when it couldn't hold on the total number of agents, it means that large number of agents can reach rapidly the boundaries

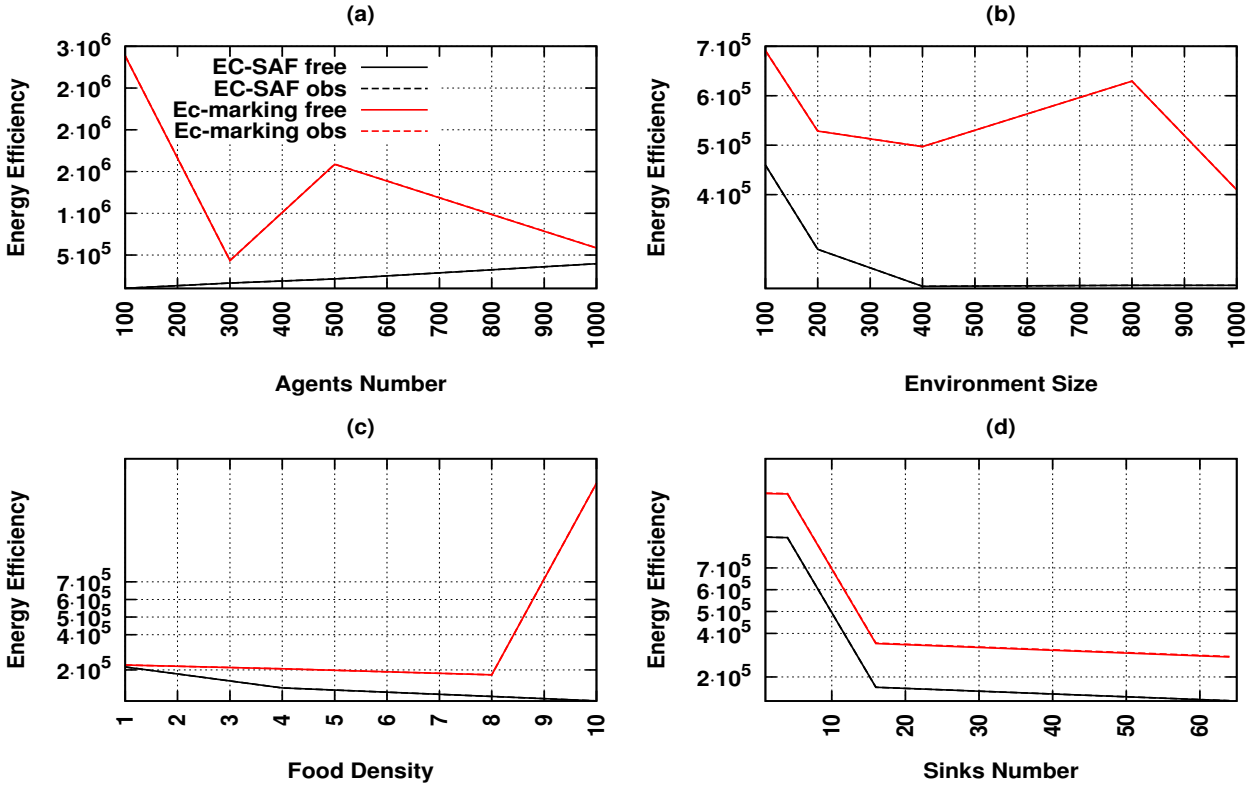


FIG. 4.4. *Simulations showing E_{eff} in world with and without obstacles when varying:* (a) agents number, (b) world size, (c) food density, (d) sink number. Where the legend in (a) is the same for the other Figs

(and stop search) even if food is already found. If the world size is sufficient, more agents will contribute at transporting food, increasing by the way the total amount of returned food, thus less E_{eff}. In Ec-marking, the E_{eff} changes overtime when world size is 800x800 cells it increases dramatically while it was decreasing with the other world sizes. Graphical representation of results is in Fig. 4.4(b).

Increasing food density helps in improving results in the two algorithms. It helps in increasing the amount of foraged food because at each increase in food locations, agents in EC-SAF will be divided between several locations, thus the paths to traverse are smaller and the energy consumed is lower (see Fig. 4.2(d)). E_{eff} in the two algorithms is very close when using one food location, it is because in Ec-marking the food is located quickly, but with increasing the food locations, results tend to decrease and converge from each other. They are higher in Ec-marking. Plots in Fig. 4.4(c) present the results obtained by the two algorithms in obstacle-free and obstacle environments.

In sub-scenario 5, when dividing the environment (by using several sinks rather than one), search space becomes smaller and paths shorter in both of the algorithms. E_{eff} is reduced, since energy consumption is less and returned food is more. In Ec-marking, E_{eff} is higher than EC-SAF (see Fig. 4.4(d)).

4.4.2. Results in Scenario 2. The growth of E_{eff} is constant overtime in both of the algorithms. In EC-SAF, *Total Energy Consumed* is less because agents avoid unnecessary moves by avoiding already visited cells, while it is higher in Ec-marking, since agents need to return to already visited cells several times in order to reach the optimal values of the Artificial Potential Field (APF) (Fig. 4.5). Since results obtained with EC-SAF in obstacle-free and obstacle environment are very close, their curves overlap in Fig. 4.5.

4.4.3. Results in Scenario 3. When agent density is fixed, the growth in total energy consumed is linear in EC-SAF algorithm. The foraging time increases which means more search time and more energy consumption

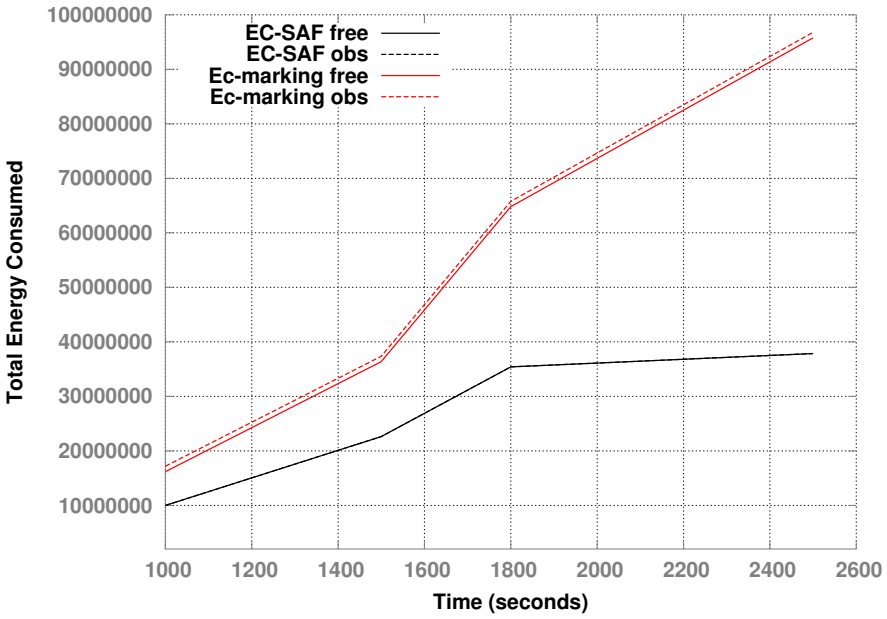


FIG. 4.5. *Total Energy Consumed overtime*

until agents reach the total amount of food. Varying both parameters (*agent number* and *world size*) at the same time does not affect the growth of E_{eff}. Table 4.5 and Fig. 4.6(a) present the evolution of E_{eff} in both algorithms overtime when fixing agent density. In Ec-marking, the growth E_{eff} is higher, it starts at decreasing when food is located. Also in this scenario, results obtained with EC-SAF (also with Ec-marking) in obstacle-free and obstacle environment are very close and their curves overlap in Figs. 4.6(b) and 4.6(c). The E_{eff} in Tables 4.5 and 4.6 is given in kilo.

TABLE 4.5
Results in scenario 3 when keeping agent density fixed to 0.01

Agent density	0.01	0.01	0.01	0.01
Agent number	100	2500	3600	6400
World size	100x100	500x500	600x600	800x800
E_{eff} **in obstacle-free world**				
EC-SAF	14	190	394	621
Ec-marking	720	8263	444643	11616
E_{eff} **in obstacle world**				
EC-SAF	15	206	464	807
Ec-marking	758	16526	444652	20202

When decreasing agent density in sub-scenario 2, E_{eff} decrease is constant and slow in EC-SAF. When decreasing agent density, we got a very low decrease in E_{eff}, since the varying parameter is world size, which doesn't influence significantly E_{eff}. While the growth of E_{eff} is inconstant in Ec-marking, it grows in linear until the value density 0.00004, it increases dramatically to higher value, and decreases after that when density is 0.0001 (see Fig. 4.6(b)).

In sub-scenario 3, in EC-SAF more agents with smaller paths provide fast foraging time and less E_{eff}. It reaches its minimum value in 0.0001 density value, over it the increase in agent number provides long paths to traverse until food location, thus large search time and more E_{eff}. In Ec-marking, the growth in E_{eff} is constant but still very high in comparison to EC-SAF (see Fig. 4.6(c)).

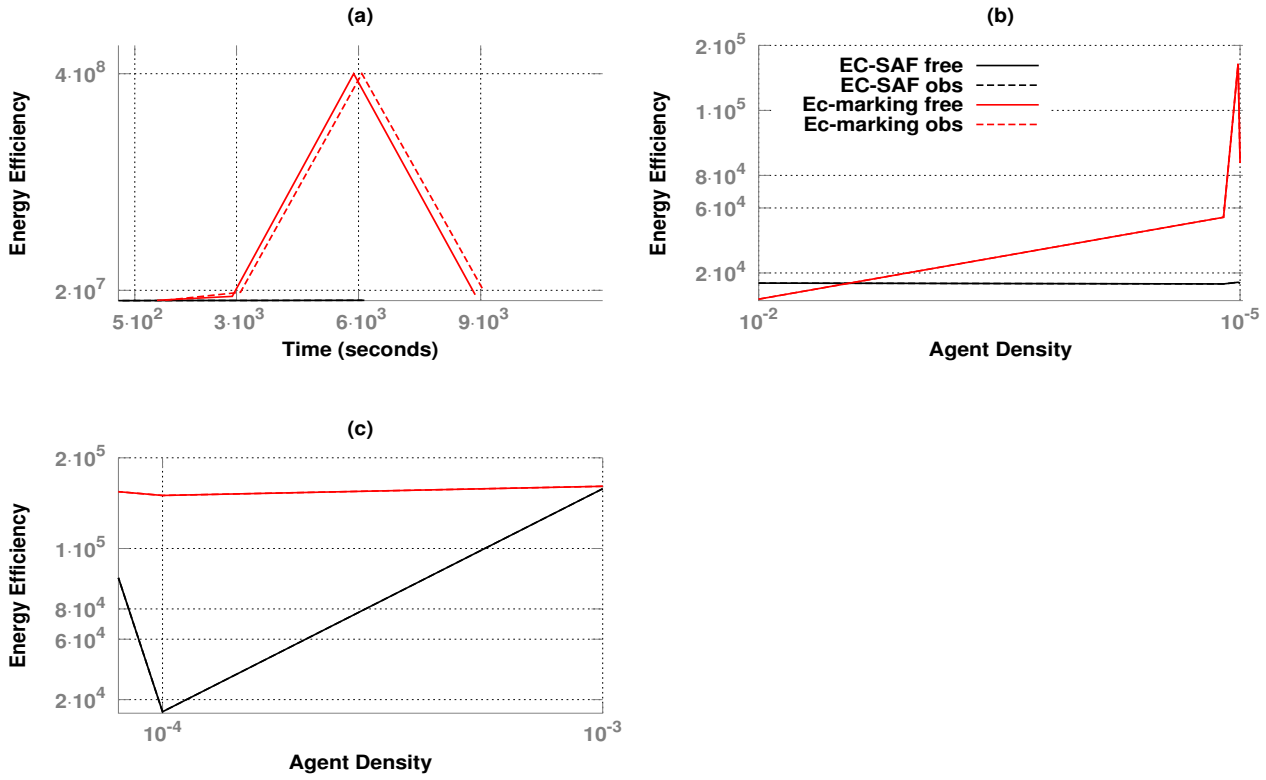


FIG. 4.6. *Simulation results of scenario 3 represents the variations in E_{eff} when:* (a) keeping Agent Density fixed, (b) decreasing Agent Density, (c) increasing Agent Density. *Where the legend in (b) is the same for the other Figs*

TABLE 4.6
Varying Agent Density

Agent density	0,01	0,001	0,0001	0.0004	0.00001	0.00006
Results without obstacle						
E_{eff} *EC-SAF*	14	159	14	13	101	14
T_{forag} *EC-SAF*	*100*	*4245*	*461*	*501*	*2943*	*775*
E_{eff} Ec-marking	4	161	149	54	158	88
T_{forag} Ec-marking	453	4232	4232	794	3948	4516
Results with obstacles						
E_{eff} *EC-SAF*	14	159	14	13	101	14
T_{forag} *EC-SAF*	*202*	*4340*	*565*	*635*	*3043*	*877*
E_{eff} Ec-marking	4	161	149	54	158	88
T_{forag} Ec-marking	658	2023	4432	105	4184	4713

The growth of E_{eff} in EC-SAF is constant when varying *Agent Density* and inconstant in Ec-marking. The EC-SAF algorithm is scalable, and it can be used for large systems with large number of agents and large world sizes. However, it presents better results when adjusting some parameters. Table 4.6 presents the obtained results in terms of E_{eff} and T_{forag} for both of the algorithms when varying *Agent Density* in obstacle-free and obstacle environment.

4.4.4. Summary of Results. Varying some parameters have trivial results and doesn't need simulations such as *agent capacity* and *food concentration*. When varying *agent capacity*, we got a decrease in foraging time, a decrease in energy consumption (since agents need less visits to food) and increase in the *Total Food Returned* (since the agent capacity is increased), thus $E_{eff}(t)$ decreases dramatically when increasing the *agent capacity*. Unfortunately, when increasing *food concentration*, agents need more visits to exhaust the food, thus more energy consumption and higher $E_{eff}(t)$. However, some parameters affect $E_{eff}(t)$, E_{eff} and must be adjusted together in order to improve the performances. *Agent number* can limit the scalability of the proposed protocol when using it in small *world sizes* with one food location. It provides large paths to traverse by agents and reduces the number of agents participating at the transport task which results in consuming more energy and reduces the number of food returned. While dividing the environment into smaller sub-spaces provides shorter paths to food, thus less energy consumption and large amount of food returned. EC-SAF protocol allows constant growth in results when used with large number of agents and large environment sizes, however, the influencing parameters need to be adjusted together. Thus, the large number of agents should be used with more than one food location in order to reduce the length of paths and to increase recruitment of agents (to transport quickly the food) and under sufficient world size that can allow all agents to participate to transport task.

5. Conclusion. This paper presents a study of the energy-aware EC-SAF algorithm performances in large-scale multi-robot foraging systems. It analyzed the parameters which can affect the *Energy Efficiency*. Different scenarios were used therefore, and we conclude that the proposed algorithm is scalable and can be used with large environmental configurations. However, it is influenced by the *agent number*, *world size*, *food density* and their adjustment together provides better performances. The shortest paths and the avoiding of unnecessary moves (revisiting already visited cells) provided by S-MASA helps in giving better results than the Ec-marking algorithm that uses pseudo random walk.

In the future, we intend to explore other environment configurations and examine other possibilities to reduce the energy consumption in EC-SAF.

REFERENCES

[1] N. El. Zoghby, V. Loscri, E. Natalizio and V. Cherfaoui, *Robot cooperation and swarm intelligence*, Wireless Sensor and Robot Networks: From Topology Control to Communication Aspects, 2014, pp. 168–201.
[2] M. Dorigo, E. Bonabeau and G. Theraulaz, *Ant algorithms and stigmergy*, Future Generation Computer Systems, 16 (2000), pp. 851–871.
[3] L. Pitonakova, R. Crowder and S. Bullock, *Understanding the role of recruitment in collective robot foraging*, MIT Press, (2014).
[4] O. Zedadra, N. Jouandeau, H. Seridi and G. Fortino, *Energy Expenditure in Multi-Agent Foraging: An Empirical Analysis*, Proceedings of the 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), 2015, pp. 1773–1778, IEEE.
[5] O. Zedadra, N. Jouandeau, H. Seridi and G. Fortino, *S-MASA: A Stigmergy Based Algorithm for Multi-Target Search*, Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Annals of Computer Science and Information Systems, 2 (2014), pp. 1477-1485.
[6] O. Zedadra, N. Jouandeau, H. Seridi and G. Fortino, *Design and Analysis of Cooperative and Non-Cooperative Stigmergy-based Models for Foraging*, Proceedings of the 19th IEEE International Conference on Computer Supported Cooperative Work in Design, 2015.
[7] Y. Zhang, *Observation-Based Proactive Communication in Multi-Agent Teamwork*, Scalable Computing: Practice and Experience, 8 (2007), pp. 63–77.
[8] F. Pasqualetti, A. Franchi and F. Bullo, *On optimal cooperative patrolling*, 49th IEEE Conference on Decision and Control (CDC), 2010, pp. 7153–7158.
[9] J. Stipes, R. Hawthorne, D. Scheidt and D. Pacifico, *Cooperative localization and mapping*, Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, 2006, pp. 596–601.
[10] A. Marjovi, J. G. Nunes, L. Marques and A. de Almeida, *Multi-robot exploration and fire searching*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 1929–1934.
[11] M. Brambilla, E. Ferrante, M. Birattari and M. Dorigo, *Swarm robotics: a review from the swarm engineering perspective*, Swarm Intelligence, 7 (2013), pp. 1–41
[12] M. Dorigo, M. Birattari and M. Brambilla, *Swarm robotics*, Scholarpedia, 9 (2014), pp. 1463
[13] S. Konur, C. Dixon and M. Fisher, *Analysing robot swarm behaviour via probabilistic model checking*, Robotics and Autonomous Systems, 60 (2012), pp. 199–213

[14] A. Saxena, C. Satsangi and A. Saxena, *Collective collaboration for optimal path formation and goal hunting through swarm robot*, 5th International Conference on Confluence The Next Generation Information Technology Summit (2014), pp. 309–312
[15] D. H. Kim, *Self-organization for multi-agent groups*, International Journal of Control Automation and Systems, 2 (2004), pp. 333–342
[16] S. Nouyan, R. Gross, M. Bonani, F. Mondada and M. Dorigo, *Teamwork in self-organized robot colonies*, IEEE Transactions on Evolutionary Computation, 13 (2009), pp. 695–711.
[17] M. Ntika, P. Kefalas and I. Stamatopoulou, *Formal modelling and simulation of a multi-agent nano-robotic drug delivery system*, Scalable Computing: Practice and Experience, 15 (2014), pp. 217-230.
[18] A. FT. Winfield,*Foraging robots*, Encyclopedia of complexity and systems science, pp. 3682–3700, 2009.
[19] K. E. Holekamp, J. E .Smith, C. C. Strelioff, R. C. Van Horn and H. E. Watts, *Society, demography and genetic structure in the spotted hyena*, Molecular Ecology, 21 (2012), pp. 613–632.
[20] P. E. Stander and S. D. Albon, *Hunting success of lions in a semi-arid environment*, Symposia of the Zoological Society of London, 1993, pp. 127–143.
[21] A. Arab, A. M. Costa-Leonardo and others, *Dynamics of foraging and recruitment behavior in the Asian subterranean termite Coptotermes gestroi (Rhinotermitidae)*, Psyche: A Journal of Entomology, 2012.
[22] T. Kuyucu, I. Tanev and K. Shimohara, *Evolutionary optimization of pheromone-based stigmergic communication*, Applications of Evolutionary Computation, 2012, pp. 63–72.
[23] N. Hoff, R. Wood and R. Nagpal, *Distributed Colony-Level Algorithm Switching for Robot Swarm Foraging*, Springer Distributed Autonomous Robotic Systems, 2013, pp. 417-430.
[24] S. P. Beeby, M. J. Tudor and NM. White, *Energy harvesting vibration sources for microsystems applications*, Measurement science and technology, 17 (2006), pp. 175.
[25] M. JB. Krieger and JB. Billeter, *The call of duty: Self-organised task allocation in a population of up to twelve mobile robots*, Robotics and Autonomous Systems, 30 (2000), pp. 65–84.
[26] T. H. Labella, M. Dorigo and J. L. Deneubourg, *Division of labor in a group of robots inspired by ants' foraging behavior*, ACM Transactions on Autonomous and Adaptive Systems (TAAS), 1 (2006), pp. 4–25.
[27] W. Liu, A. FT. Winfield, J. Sa, J. Chen and L. Dou, *Towards energy optimization: Emergent task allocation in a swarm of foraging robots*, Adaptive behavior, 15 (2007), pp. 289–305.
[28] A. Campo and M. Dorigo, *Efficient multi-foraging in swarm robotics*, Advances in Artificial Life, 2007, pp. 696–705
[29] J. H. Lee, C. W. Ahn and J. An, *A honey bee swarm-inspired cooperation algorithm for foraging swarm robots: An empirical analysis*, IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 2013, pp. 489–493
[30] A. Couture-Beiland R. T. Vaughan, *Adaptive mobile charging stations for multi-robot systems*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2009, pp. 1363–1368
[31] S. Kernbach and O. Kernbach, *Collective energy homeostasis in a large-scale microrobotic swarm*, Robotics and Autonomous Systems, 59 (2011), pp. 1090–1101
[32] A. F. Winfield, S. Kernbach and T. Schmickl, *Collective foraging: cleaning, energy harvesting and trophallaxis*, Handbook of Collective Robotics: Fundamentals and Challenges, Pan Stanford Publishing, Singapore, 2011, pp. 257–300
[33] O. Simonin, F. Charpillet and E. Thierry, *Revisiting wavefront construction with collective agents: an approach to foraging*, Swarm Intelligence, (2014), pp. 113–138.
[34] O. Zedadra, N. Jouandeau, H. Seridi and G. Fortino, *A Distributed Foraging Algorithm Based on Artificial Potential Field*, Proceedings of the 12th IEEE International Symposium on Programming and Systems (ISPS), 2015, pp. 1–6.
[35] R. A. Brooks, *A robust layered control system for a mobile robot*, Journal of Robotics and Automation, 2 (1986), pp. 14–23.
[36] J. Barraquand, B. Langlois and J. -C. Latombe, *Numerical potential field techniques for robot path planning*, IEEE Transactions on Systems, Man and Cybernetics, 22 (1992), pp. 224–241.
[37] T. Balch, *Grid-based navigation for mobile robots*, The Robotics Practitioner, 2 (1996), pp. 6–11.
[38] Y. P. Yean and RM. K. Chetty, *An efficient grid based navigation of wheeled mobile robots based on visual perception*, Trends in Intelligent Robotics, Automation, and Manufacturing, 2012, pp. 128–135.
[39] B. Lau, C. Sprunk and W. Burgard, *Efficient grid-based spatial representations for robot navigation in dynamic environments*, Robotics and Autonomous Systems, 61 (2013), pp. 1116–1130.
[40] U. Wilensky, NetLogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.

Edited by: Dana Petcu
Received: August 27, 2015
Accepted: October 4, 2015