

Bee System with inhibition Pheromones

Nyree Lemmens¹, Steven de Jong², Karl Tuyls², and Ann Nowé¹

¹ Computational Modeling Lab (CoMo), Vrije Universiteit Brussel, Belgium

² Adaptive Agents Group, MICC/IKAT, Maastricht University, The Netherlands

Summary. In previous work, we developed a foraging algorithm inspired by the behavior of biological bees. In relatively unobstructed environments, this algorithm has been shown to outperform an ant-inspired algorithm in terms of efficiency and scalability. However, due to its low adaptability outside of the hive, our bee-inspired algorithm displays weaker performance in more obstructed environments due to its nature to always use a straight, direct path to its destination without taking obstacles into account. In this paper, we present Bee System with inhibition Pheromones, a new, hybrid algorithm based on the recruitment and navigation strategies of bees and extended with inhibition pheromones to enhance adaptability. Moreover, wall following has been implemented to improve obstacle avoidance. We show that the hybrid algorithm truly combines the ‘best of both worlds’ and manages to outperform both our ant-inspired algorithm as our bee-inspired algorithm in a variety of experimental settings.

1.1 Introduction

In this paper we introduce a hybrid swarm intelligence algorithm named Bee System with inhibition Pheromones (BSP), combining the efficiency and scalability of our previously introduced bee-inspired algorithm [1, 2, 3] with the learning capabilities of ant-inspired algorithms. More precisely, we extend our bee algorithm with inhibition pheromones to enhance its adaptability. Moreover, we improve obstacle avoidance by adding wall following behavior to the algorithm. The algorithm is applied to the foraging domain and is evaluated on different dynamic environments with various obstructions.

Foraging is a difficult optimization problem in the domain of multi-agent systems. The problem entails that a group of agents has to gather objects (food) in a complex, potentially dynamic environment [4]. Traditionally, foraging algorithms were based on multi-agent planning; however, multi-agent planning does not scale well with a growing number of agents or a growing environment [5]. More recently therefore, researchers have been applying swarm intelligence techniques, most notably ant systems, inspired by the biological behavior of ants [6]. The ants’ behavior essentially consists of two parts: a recruitment strategy and a navigation strategy. A recruitment strategy is used by the ants to distribute knowledge on the locations

of food sources to other members of the colony. A navigation strategy is used to efficiently navigate to and from food sources. As is commonly known, ants use pheromone trails for both recruitment and navigation.

In previous work, we investigated the possibilities of developing a swarm intelligence algorithm based on the behavior of honeybees [1, 2]. More precisely, we developed an algorithm that combined both the recruitment and the navigation strategies observed in biological honeybees; existing work focussed only on one of these strategies. We compared the efficiency of our bee-inspired algorithm with an ant-inspired algorithm (with features of ACS and MMAS [6]). Our comparison showed that our bee-inspired algorithm clearly outperforms the ant-inspired algorithm in a number of different, relatively unobstructed, foraging environments. Unlike the ant-inspired algorithm however, our bee-inspired algorithm is unable to learn from any mistakes (i.e., bumping into obstacles) when outside of the hive. This is due to its nature to always use a straight, direct path to its destination without taking obstacles into account. Thus the bee-inspired algorithm is clearly less adaptive than the ant-inspired algorithm.¹

Both our bee-inspired algorithm and the ant-inspired algorithm thus show some specific shortcomings; in the ant-inspired algorithm, optimal paths may take considerable time to emerge and in our bee-inspired algorithm, agents tend to get stuck behind obstacles (for instance in the Deneubourg Bridge scenario [7, 3]).

In our current work, we present a hybrid approach to the task of foraging, combining the best properties of both the ant-inspired algorithm and our bee-inspired algorithm. In this improved algorithm, the agents still perform the same recruitment and navigation strategies as in the original algorithm. However, we introduce two additional features: (1) agents exhibit the wall-following behavior observed in various insects [8] and (2) agents can place inhibition pheromones in cells that can be considered undesirable, for instance cells that lead to a dead end. In other words, this improved algorithm integrates some features of ant-inspired algorithms into a bee-inspired algorithm. In a set of experiments, we show that BSP outperforms both our bee-inspired algorithm and the ant-inspired algorithm in relatively unobstructed environments as well as more constrained environments. At the same time, the algorithm requires only slightly more computation time than our bee-inspired algorithm. Thus, BSP constitutes a good alternative to existing techniques in the domain of foraging.

The remainder of this paper is structured as follows. First, we briefly discuss biological bee behavior. Then, we give an overview of related work in the area of swarm intelligence and bee-inspired algorithms. We continue with a brief look at the characteristics of foraging problems. Next, we present our new algorithm, BSP, and a set of experiments. Finally, we conclude and look at future research.

¹ We have to indicate that our experiments take place in a 2D environment. As a consequence, the artificial bees are unable to fly over encountered obstacles. Although this would mean that we take the bee behaviour out of its context, we have to indicate that ground-bound desert ants (i.e., *Cataglyphis Fortis*) also employ bee navigation behaviour.

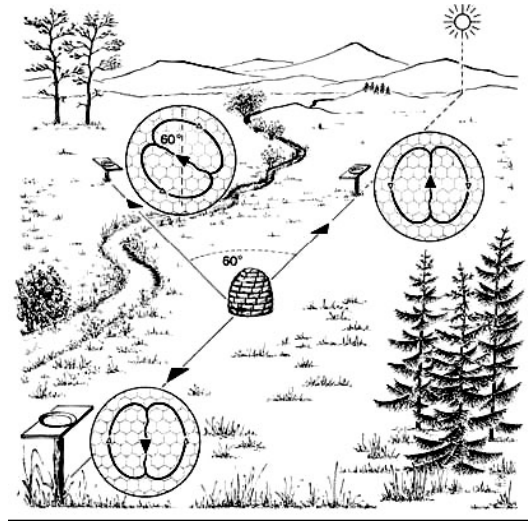


Fig. 1.1. Distance and direction by wagging dance. Wagging straight up on the vertical comb indicates a food source which is located at an azimuthal angle of 0° while wagging straight down indicates a food source located at an azimuthal angle of 180° . Figure is taken from [13].

1.2 Biological Bee Behavior

Honeybee foraging behavior consists of two types of behavior, i.e., recruitment and navigation behavior. In order to recruit other colony members for food sources, honeybees inform their nest mates of the distance and direction of these food sources by means of a wagging dance performed on the vertical combs in the hive [9]. This dance (i.e., the bee language) consists of a series of alternating left-hand and right-hand loops, interspersed by a segment in which the bee waggles her abdomen from side to side. The duration of the waggle phase is a measure of the distance to the food, and the angle between the sun and the axis of this waggle segment on the vertical comb represents the azimuthal angle between the sun and the direction in which the recruit should fly to find the target [9, 10, 11] (see Figure 1.1). The ‘advertisement’ for a food source can be adopted by other members of the colony. The decision mechanism for adopting an ‘advertised’ food-source location by a potential recruit, is not completely known. It is considered that the recruitment amongst bees is a function of the quality of the food source [12].

Different species of social insects, such as honeybees and desert ants, make use of non-pheromone-based navigation. Non-pheromone-based navigation mainly consists of Path Integration (PI), which is the continuous update of a vector by integrating all angles steered and all distances covered [14]. A PI vector represents the insect’s knowledge of direction and distance towards its destination. To construct a PI vector, the insect does not use a mathematical vector summation as a human does, but employs a computationally simple approximation [15]. Using this approximation, the insect is able to return to its destination directly. More precisely, when

the path is unobstructed, the insect solves the problem optimally. However, when the path is obstructed, the insect has to fall back on other strategies such as exploration or landmark navigation [16, 17] to solve the problem. Obviously, bees are able to fly and when they encounter an obstacle they can mostly choose to fly over it. However, even if the path is unobstructed, bees tend to navigate over the entire path using landmarks. Such navigation behavior decreases navigation errors and ensures robustness.

1.3 Related Work

Although ant and bee foraging strategies differ considerably, both species solve the foraging problem efficiently. In the field of Artificial Intelligence, researchers have become inspired by the behavior of social insects, since the problems these insects cope with are similar to optimization problems that humans wish to solve efficiently, such as the Shortest Path Problem.

Most work in the field of swarm intelligence has concentrated on pheromone-based algorithms, which are inspired by the behavior of ants. For an overview, we refer to [6]. In summary, ants deposit pheromone on the path they take during travel. Using this trail, they are able to navigate towards their nest or food. Ants employ an indirect recruitment strategy by accumulating pheromone trails. When a trail is strong enough, other ants are attracted to it and will follow this trail towards a destination. More precisely, the more ants follow a trail, the more that trail becomes attractive for being followed. This is known as an autocatalytic process. Since long paths take more time to traverse, it will require more ants to sustain a strong path. As a consequence, short paths will eventually be preferred. Pheromone-based algorithms are already used to address various problems, such as the Traveling Salesman Problem, successfully [6].

Non-pheromone-based algorithms are inspired by the behavior of (mainly) bees and do not use pheromones to navigate through unfamiliar worlds. Instead, for navigation, they use Path Integration. Artificial bees are able to compute their present location from their past trajectory continuously and, as a consequence, can return to their starting point by choosing the direct route rather than retracing their outbound trajectory [14, 15]. For recruitment, artificial bees employ a direct strategy by ‘dancing’ in the nest. Their dance communicates distance and direction towards a destination [9]. Non-pheromone-based algorithms are less extensively studied than pheromone-based algorithms. For instance, [18, 19, 20, 21, 22] all present bee-inspired algorithms which pose solutions to different types of problems by employing bee recruitment behavior. In [14], the navigation behavior of bees is investigated and applied in a robot. However, these algorithms all use only one aspect of bee behavior, i.e., the recruitment behavior or navigation behavior respectively.

In [1, 2, 3], we introduced an algorithm combining both bee recruitment and navigation behavior. We compared this algorithm with an ant-inspired algorithm, focussing on the efficiency, scalability and adaptability of the algorithms. We found that our bee-inspired algorithm was approximately three times more efficient than the

ant-inspired algorithm in relatively unobstructed environments, i.e., our bee-inspired algorithm managed to gather all the food present in the environment in three times less time steps than the ant-inspired algorithm. Partially because of this, our bee-inspired algorithm was also more scalable than the ant-inspired algorithm, i.e., with the ant-inspired algorithm, the whole solution took a longer time to compute than with our bee-inspired algorithm. Unfortunately, our bee-inspired algorithm turned out to be less adaptive than the ant-inspired algorithm; if an environment contains many obstacles, the ant-inspired algorithm is able to learn from following erroneous paths, but our bee-inspired algorithm is not. As a result, in such an environment, the solution obtained by our bee-inspired algorithm is inferior to that found by the ant-inspired algorithm.

1.4 Characteristics of Foraging Environments

In this section, we briefly summarize the characteristics of foraging environments. Usually, foraging takes place in a large environment. In our case, this environment is divided in a square grid with a four-neighbour topology, varying in size from 10x10 to 40x40 cells. Per time step, agents can make one move from a certain cell to another. Depending on the problem at hand, this environment can have various other properties, e.g.:

- *Number of constraints.* Some environments have few or no obstacles and thus allow for unconstrained movement. In other environments, movement is constrained, with the most extreme case being a labyrinth.
- *Food source dynamics.* An environment is said to be *static* with respect to the food sources if the locations of these sources do not change in time; note that food source locations may run out of food on the long run. On the other hand, the environment is *dynamic* if food sources can suddenly appear or be moved to a new location.
- *Obstacle dynamics.* An environment is said to be *static* with respect to obstacles if obstacles cannot suddenly appear or disappear. Otherwise, the environment is *dynamic* in this respect.

As has been mentioned earlier, especially a growing number of constraints turned out to be a problem for the bee-inspired algorithm we developed [3]. For instance, in the Deneubourg Bridge experiment [7], adding a single constraint makes the problem at hand unsolvable for our bee-inspired algorithm [3].

1.5 The BSP algorithm

In this section, we will describe the algorithm developed. First, for the sake of clarity, we give an overview of the original bee-inspired algorithm. Then, for convenience, we again indicate two weak points of this algorithm. Finally, we propose a set of extensions to the original bee-inspired algorithm, leading to the hybrid BSP algorithm.

1.5.1 Original bee-inspired algorithm

The bee algorithm implements both recruitment and navigation behavior. *Recruitment behavior* is implemented in analogy with biological bees' dance behavior. Agents (artificial bees) share information on previous search experience (i.e., the direction and distance toward a certain food source) only when they are in the hive. Agents in the hive can then decide whether to exploit previous search experience obtained by other agents in the hive, or to exploit their own search experience, if available. Biological bees use a (still) unknown decision mechanism to decide whether to exploit another bee's experience. In our bee-inspired algorithm, the decision is based on distance assessment; an agent will exploit another agent's experience if this experience indicates food sources at a shorter distance from the hive than the food source currently known by the agent. The *navigation behavior* used in our bee-inspired algorithm either exploits previous search experience (of the agent itself or of another agent in the hive) or lets the agents explore the world using an exploration strategy similar to a Lévy flight [23]. Exploiting previous search experience is guided by a PI vector that agents either have constructed themselves or have adopted from another agent in the hive.

The general structure of our bee-inspired algorithm is quite similar to that of algorithms in ACO [6]. It implements both recruitment and navigation behavior and consists of three functions.

First, *ManageBeesActivity()* handles agents' activity based on their internal state. Each agent is in one of six internal states. In each state a specific behavior is performed. Agent state 'AtHome' indicates that the agent is located at the hive. While in this state, the agent determines to which new state it will go. Agent state 'StayAtHome' also indicates that the agent is located at the hive. However, while in this state it will remain there unless there is previous search experience available to exploit. In the latter case, the agent will leave the hive to exploit the previous search experience. Agent state 'Exploitation' indicates that the agent is exploiting previous search experience. Previous search experience is represented by a PI vector indicating a food source. The agent determines which cell to move to in order to match the PI vector indicating the food source. Agent state 'Exploration' indicates that the agent is exploring its environment in search for food. Agent state 'HeadHome' indicates that the agent is heading home without carrying any food. The agent reaches home by following its Homing Vector (HV). The HV is a PI vector indicating the hive. From the moment an agent starts its foraging trip, this HV is continuously calculated for each agent. Agent state 'CarryingFood' indicates that the agent has found food and that it is carrying the food back towards the hive. The agent's return path depends on the same HV as with agent state 'HeadHome'. For more technical details on how state changes occur we refer to [1, 2].

Second, *CalculateVectors()* is used to compute the PI vectors for each agent, i.e., the HV and possibly the PI vector indicating the food source. A PI vector essentially consists of two values, one indicating the direction and the other indicating the distance. A PI vector is always calculated with respect to the previous one. In order to calculate the new homing distance, we use the cosine rule and rewrite it to:

$$b = \sqrt{a^2 + c^2 - 2ac \times \cos\beta} \quad (1.1)$$

In Equation 1.1, a represents the distance travelled since the last turn was made, c the old homing distance, and b the new homing distance. β is the angle turned with respect to the old homing angle. Using Equation 1.1 we can now calculate α (the angle used for adjusting the old homing angle), once again by using the cosine rule.

$$\alpha = \arccos\left(\frac{a^2 - b^2 - c^2}{-2bc}\right) \quad (1.2)$$

Values obtained by Equation 1.1 and Equation 1.2 are used to construct the new PI vector.

Third, *DaemonActions()* can be used to implement centralized actions which cannot be performed by single agents, such as collection of global information which can be used to decide whether it is useful to let an agent dance. In our bee-inspired algorithm, *DaemonActions()* is not used.

1.5.2 Weak points of the original bee-inspired algorithm

As has been mentioned earlier, our bee-inspired algorithm works very well in relatively unobstructed environments; foraging tasks are typically completed in about three times less time steps than with the ant-inspired algorithm [1]. However, as environments become more obstructed, the performance of the algorithm drops [2, 3]. With certain environments, no artificial bee is able to reach the hive anymore. This is not surprising; after all, the algorithm is based on the behavior of flying insects, which are typically not really bothered by obstacles. If such insects do fly into an obstacle, they either fly higher or explore for a while and then continue following their path. Since our simulation environment is two-dimensional, obviously our artificial bees can only perform the exploration procedure whenever they bump into an obstacle while following their PI vector. However, if the exploration phase does not take long enough or the obstacle is large, the agent will once again bump into the same obstacle. More sophisticated methods for obstacle avoidance can help to solve this problem.

A second weak point of our bee-inspired algorithm is that the agents do not learn from mistakes; if one agent has bumped into a wall when following its PI vector back to the hive, other agents traveling the same path will make exactly the same error. In fact, even the agent that made the error will happily do so again. This problem can be dealt with by introducing the capacity to learn into the agents and/or the environment. The latter possibility would add extra information inside the world that could be exploited, in analogy to ants which drop pheromone in the world which represents the ants' memory.

Obviously, both in the case of more advanced obstacle avoidance and implementing learning techniques, special care must be taken to ensure that the resulting algorithm still displays the benefits of the original bee-inspired algorithm.

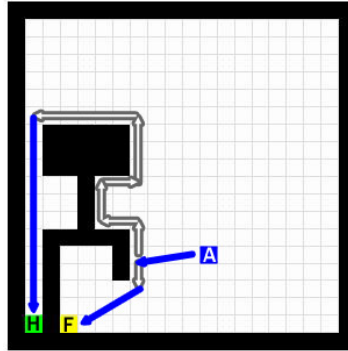


Fig. 1.2. Illustration of wall-following behavior. The open arrows are movements caused by following the wall. The solid arrows are movements caused by agent A's PI-vector, either moving towards the hive H or a food source F.

1.5.3 Bee System with inhibition Pheromones

In order to improve our bee-inspired algorithm, we propose two extensions to this algorithm, both of which are inspired by ants. Obviously, special care must be taken to ensure that the resulting algorithm still displays the benefits of the original bee-inspired algorithm. We call the resulting algorithm BSP, for Bee System with inhibition Pheromones.

The first extension improves the obstacle avoidance capabilities of the agents in a relatively simple way. Whenever an agent bounces into an obstacle while following a PI vector, it randomly selects a direction (left or right) and then starts following the contours of the obstacle in that direction, until following the PI vector becomes possible again. This behavior is illustrated in Figure 1.2.

The second extension introduces learning into the algorithm. More precisely, agents can deposit a certain quantity of inhibition pheromone at a certain location. In a static environment, this pheromone can be permanent; in a dynamic environment, the pheromone can slowly evaporate, ensuring that the agents adapt to changes in the environment, such as obstacles being removed. Moreover, in highly dynamical worlds, all the pheromone can be instantly forgotten with a certain, low probability. The pheromone deposited at a specific location is a sign to all exploiting agents (i.e., agents following a PI vector) not to move to this location. Depending on whether the environment is static or dynamic, exploring agents can be instructed to stay out of marked locations as well, or to completely ignore the pheromone marks, respectively; in a static environment, locations that were marked once are certainly good to stay out of, whereas in a dynamic environment, an item of food may appear in an area that was previously marked. In this case, the agent finding the food 'eats' the pheromone that is still present on its way from the food source to the hive. In our experiments, exploring agents are allowed to explore marked locations, i.e., we do not assume an environment to be static.

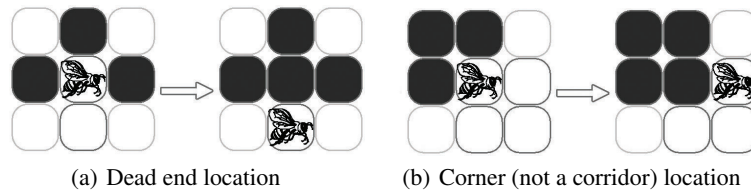


Fig. 1.3. Situations in which pheromone is deposited.

We defined two situations in which agents actually deposit pheromones. The two situations are illustrated in Figure 3(a) and 3(b). First, whenever an agent is surrounded by three walls (where walls include locations where pheromone was already deposited), and there is no food or hive present there, this location clearly is a dead end (see Figure 3(a)). As a consequence, it should no longer be visited and therefore receives inhibition pheromone. Second, the same applies whenever an agent is standing in a corner, but not in a corridor, and there is no food or hive present in this corner (see Figure 3(b)).

Both extensions can be expected to increase the performance of the BSP algorithm while keeping its scalability acceptable; following walls and depositing pheromones definitely increases the ability of the algorithm to navigate back home (or towards food), without requiring too much additional computation. In the most extreme case, for instance in a world with a large number of obstacles, there is pheromone almost everywhere in the environment, leading to performance and computational demands similar to the ant-inspired algorithm; however, pheromones are much less often deposited and updated in BSP.

1.6 Experiments and Results

In this section, we discuss our experimental set-up, i.e., the environments experimented on, their characteristics, and the settings used for each algorithm under study (i.e., the ant-inspired algorithm, the bee-inspired algorithm, and BSP, respectively). Next, we provide the results of our experiments. All experiments have been conducted in the BeeHave tool, which was developed previously to conduct comparative experiments between the ant-inspired algorithm and the bee-inspired algorithm [1, 2, 3]. The tool is now extended with the hybrid algorithm BSP. A screenshot of BeeHave is displayed in Figure 1.4.²

1.6.1 Experimental Set-up

In order to compare the three algorithms under study under various circumstances, we developed a set of three experiments. We will discuss these experiments here, including an overview of the characteristics of the environments used and the settings

² This tool is available for download from:
http://como.vub.ac.be/newsite/doku.php?id=members:nyree_lemmens

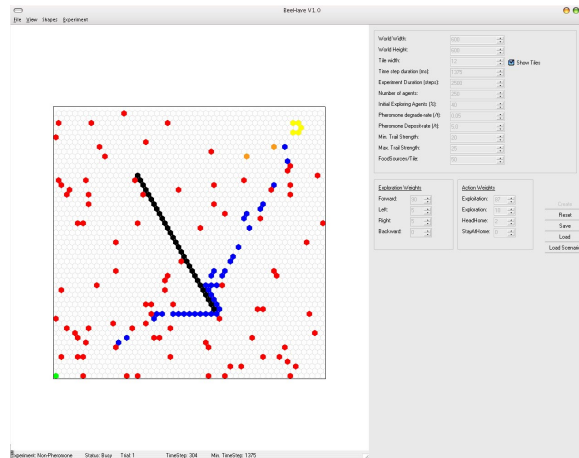


Fig. 1.4. The BeeHave tool.

used for the algorithms. For clarity, we again indicate that the experiments are done in a two-dimensional simulation. This does not take the bee-inspired algorithms out of context as the same behaviour can be found in ground-bound insects, such as the desert ant.

In every experiment, we simulate using 50 and 100 agents. Initially, 50% of the agents are exploring. Every food source contains 200 items of food; every agent can carry one such item at a time. The experiments are terminated after all the food has been gathered, or after 2000 time steps (whichever comes first) to save on simulation time and to prevent deadlocks (e.g., a deadlock can occur when using the bee-inspired algorithm. Due to the algorithm's nature all agents could get stuck behind an obstacle.) Every experiment is repeated 30 times to compensate for the inherent randomness in the algorithms. The environments are selected as follows and illustrated in Figure 1.5.

E1.1: Unobstructed environment. As a basic experiment, we run the algorithms in a static environment with no constraints. The environment consists of 10x10 cells, of which the outer edge consists of walls. There is a hive in the bottom left corner and a single food source in the top right. Since our earlier work [1, 2, 3] has already led to the conclusion that our bee-inspired algorithm is much faster than the ant-inspired algorithm, we do not include the ant-inspired algorithm in this comparison.

E1.2: Unobstructed environment. An identical set-up is used with 40x40 cells. The task is likely to be too complicated to finish within 2000 time steps by any of the algorithms. Using this experiment however, we can determine the scalability of the algorithms with respect to a growing environment size.

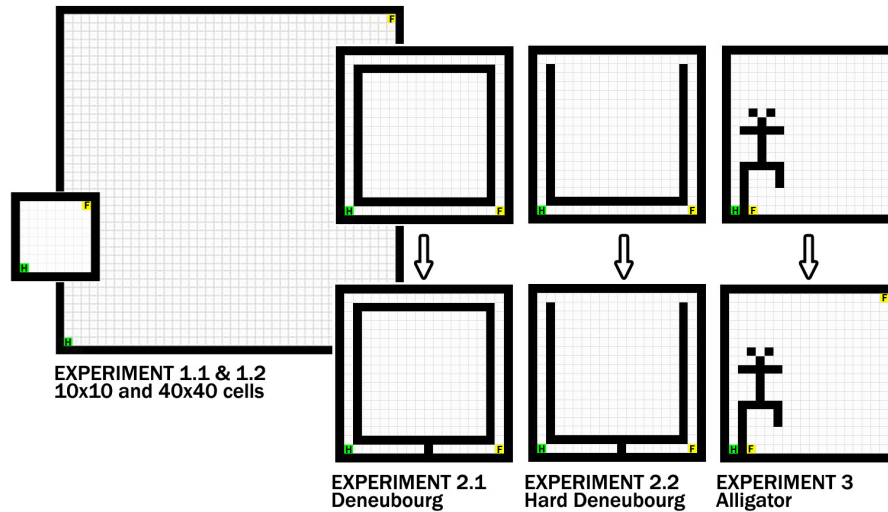


Fig. 1.5. The environments developed for the experiments. The hive is indicated with an H, the food source(s) with an F.

E2.1: Deneubourg Bridge environment. Our second experiment is the Deneubourg Bridge [7], a highly constrained environment with obstacle dynamics. The size of the environment is set to 20x20 cells, with a short (18 cells) and a long (42 cells) path between the hive and the food source. The experiment consists of three phases. Each phase has a maximum duration of 2000 time steps. The first phase presents the Deneubourg Bridge with both paths open. After the first phase is terminated, an obstacle is placed in the short path and the food source is refilled to create the second phase. Once again after the phase is terminated, the obstacle is removed and the food source is refilled to create the third phase. Finally, we wait until the experiment terminates. We then measure the average number of time steps needed over the three phases.

E2.2: Hard Deneubourg Bridge environment. An identical set-up to E2.1 is used in a modified, less constrained environment where the long path has a problematic ‘trap’ area.

E3: Alligator environment. This environment once again consists of 20x20 cells and includes food source dynamics and a complicated constraint. The experiment consists of two phases. Each phase has a maximum duration of 2000 time steps. In the first phase, the initial food source is located directly next to the hive, but the two locations are separated by a large, irregular obstacle. After the first phase is terminated, a new food source appears in the upper right corner to create the second phase. Depending on whether the original food source has been emptied or not, there are

Efficiency										
Algorithm	Experiment 1.1		Experiment 1.2		Experiment 2.1		Experiment 2.2		Experiment 3	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var	Mean	Var
Using 50 agents										
Ant	-	-	-	-	252.0	13.9	628.9	189.4	2000.0	0.0
Bee	602.6	97.6	2000.0	0.0	175.9	3.6	707.6	106.5	2000.0	0.0
BSP	418.8	129.6	2000.0	0.0	170.5	7.1	470.9	189.2	1884.4	188.4
Using 100 agents										
Ant	-	-	-	-	144.5	6.6	394.2	129.4	1977.6	100.2
Bee	264.2	29.7	1953.8	206.6	95.6	3.4	719.4	148.8	2000.0	0.0
BSP	191.4	39.7	1953.8	206.6	99.9	4.7	213.6	20.9	1387.5	283.6

Table 1.1. The efficiency of the three algorithms under study, i.e., the number of time steps used to complete the experiments at hand; both average and variance are displayed. In every experiment, the maximum possible number is 2000.

now either one or two different food sources. Once again, we wait until the experiment is terminated. We then measure the average number of time steps needed over the two phases.

In each experiment, the following settings are selected for the algorithms. For the ant-inspired algorithm, we select an evaporation of 1% per time step, 2 units of pheromone deposited per time step when carrying food, maximal trail strength of 10, minimal trail strength for a trail to be followed of 5. The two latter parameters are set empirically and increase the fairness of the comparison. For our bee-inspired algorithm, no additional settings were needed. For BSP, 1 unit of pheromone is deposited per time step in case of inhibition situations. The maximal pheromone amount in a cell is set to 1. Only exploiting agents can deposit inhibition pheromones (except in E1.1 and E1.2, where exploring agents can also deposit pheromones; BSP becomes exactly the same algorithm as our bee-inspired algorithm without this feature in completely unobstructed worlds). Exploring agents may enter inhibition cells, and if they find food in an area filled with pheromones, they can ‘eat’ the pheromone on their way back. In E3, we obtain the best performance by ‘forgetting’ all the pheromones with a probability of 0.004 per time step.

1.6.2 Results

We compare the performance of the algorithms under study with respect to their efficiency, scalability and adaptability. Efficiency can be measured by determining the number of time steps the algorithms required to gather all the food present in the environment. Scalability relates to the computation time required to perform this task. Adaptability is not directly measurable in our settings; however, an algorithm that is not able to learn from its mistakes will perform (a lot) worse than an algorithm that can.

Scalability										
Algorithm	Experiment 1.1		Experiment 1.2		Experiment 2.1		Experiment 2.2		Experiment 3	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var	Mean	Var
Using 50 agents										
Ant	-	-	-	-	93.9	3.4	96.3	2.6	85.7	2.3
Bee	16.0	0.2	17.2	1.5	17.3	0.5	16.6	0.4	21.8	0.2
BSP	18.5	0.7	17.2	1.5	18.6	0.4	17.7	0.4	20.3	0.6
Using 100 agents										
Ant	-	-	-	-	186.9	4.5	192.0	3.7	169.8	2.3
Bee	34.0	1.4	36.2	3.3	34.6	1.1	31.7	0.9	56.8	0.5
BSP	39.0	1.5	36.2	3.3	36.4	1.0	34.7	1.2	39.5	1.6

Table 1.2. The scalability of the three algorithms, i.e., the required computation time per time step in milliseconds; average and variance are displayed.

Efficiency. We report our results on the efficiency of the three algorithms in Table 1.1. In E1.1, we see that BSP manages to improve on the results of our bee-inspired algorithm. As has been mentioned earlier, our bee-inspired algorithm already constituted an improvement over the ant-inspired algorithm of approximately a factor three with respect to overall efficiency. In E1.2, the algorithms usually fail to complete the task at hand within 2000 time steps, as expected. In E2.1, our bee-inspired algorithm and BSP perform comparably well, and definitely better than the ant-inspired algorithm. The reason that our bee-inspired algorithm works here, and was reported not to work in [3], is that we used a different problem representation in earlier work. As a consequence, agents could make circular movements around the hive or the food source when returning from the blocked passage. In the current problem representation, this is no longer possible, and since agents usually prefer any move over turning back, they are able to find the alternative, longer path. In contrast, in E2.2, our bee-inspired algorithm fails to find a solution after the shortest path has been blocked. The ant-inspired algorithm does slightly better, leading to a lower average number of time steps for the whole task. BSP is by far the best-performing algorithm once again; observing it during each of the three phases of the experiments, we see that it quickly finds a good path between the hive and the food source. In E3, our bee-inspired algorithm does not find any solution. The ant-inspired algorithm manages to establish a path, but does so only just before the experiment runs out of time. Meanwhile, BSP has usually already retrieved all the food, especially with 100 agents.

Scalability. We measured the total time required to run the algorithms in the five different experiments and divided this through the number of time steps used. Results are summarized in Table 1.2. In every experiment, the ant-inspired algorithm is much slower than the other two algorithms. BSP is only slightly slower than the bee-inspired algorithm on the computation-time-per-time-step measure, even in E1.2. In E3, it is actually faster than our bee-inspired algorithm, which is caused by the fact that agents in our bee-inspired algorithm get stuck while trying to exploit their PI

vector; by implementation, exploiting agents require more computation than exploring agents. When we consider the number of time steps needed by our bee-inspired algorithm and BSP it is clear that BSP usually finds a solution using less computation time than our bee-inspired algorithm.

Adaptability. As has been mentioned earlier, adaptability is not easily measurable. However, by observing the behavior of the algorithms during the experiments and by looking at their efficiency, we may conclude that the added, ant-inspired functionality in BSP made the algorithm as adaptive as the ant-inspired algorithm; our bee-inspired algorithm, in contrast, is only adaptive with respect to changing food sources by ‘advertisement’ in the hive. It lacks adaptiveness outside of the hive.

1.7 Conclusion & Future Work

Observing our results, we may conclude that our proposed algorithm BSP indeed is a valuable addition to the currently existing collection of insect-inspired foraging algorithms. In unobstructed environments, it performs even better than a pure bee algorithm (our bee-inspired algorithm, which in turn significantly outperforms the ant-inspired algorithm). In more obstructed environments and environments in which obstacles or food can suddenly appear, the algorithm performs significantly better than a pure ant algorithm (the ant-inspired algorithm, which in turn outperforms our bee-inspired algorithm). The required computation time is only slightly higher than that of our bee-inspired algorithm, and drastically lower than that of the ant-inspired algorithm. In summary, the algorithm truly combines the ‘best of both worlds’.

The performance of the BSP algorithm is not yet optimal. In future work, we plan to optimize the algorithm further by addressing various minor issues. We will mention the two most important ones here. First, agents who are ‘eating’ inhibition pheromone obviously cannot learn from their mistakes. Second, agents can accidentally create a pheromone field around other agents if these other agents just picked up the last food from a food source. As a result, simulations sometimes do not terminate before the 2000-time-step limit. Another line of research will be devoted to implementing landmark navigation, as seen in biological bees. It will be interesting to determine how the bee-inspired algorithm with added landmarks performs in comparison to BSP and whether BSP might also benefit from the addition of landmark navigation.

References

1. Lemmens, N.: To bee or not to bee: A comparative study in swarm intelligence. Master’s thesis, Maastricht University, The Netherlands (2006)
2. Lemmens, N., de Jong, S., Tuyls, K., Nowé, A.: Bee behaviour in multi-agent systems: a bee foraging algorithm. In: Proceedings of the 7th ALAMAS Symposium. (2007) pp.126–138

3. Lemmens, N., de Jong, S., Tuyls, K., Nowé, A.: A Bee Algorithm for Multi-Agent Systems: Recruitment and Navigation Combined. In: Proceedings of ALAg, an AAMAS workshop. (2007)
4. Stephens, D.W., Krebs, J.R.: Foraging theory. Princeton University Press, Princeton NJ (1986)
5. de Weerd, M., ter Mors, A., Witteveen, C.: Multi-agent Planning: An introduction to planning and coordination. (2005) 1–32
6. Dorigo, M., Stützle, T.: The ant colony optimization metaheuristic: algorithms, applications, and advances. Technical report, Université Libre de Bruxelles (2000)
7. Deneubourg, J., Aron, S., Goss, S., Pasteels, J.: The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behaviour* **3** (1990) 159–168
8. Cowan, N.J., Lee, J., Full, R.J.: Task-Level Control of a Rapid Wall Following in the American Cockroach. *Journal of Experimental Biology* **209** (2006) 1617–1629
9. von Frisch, K.: The dance language and orientation of bees. Harvard University Press, Cambridge, Massachusetts (1967)
10. Michelsen, A., Andersen, B., Storm, J., Kirchner, W., Lindauer, M.: How honeybees perceive communication dances, studied by means of a mechanical model. *Behavioral Ecology and Sociobiology* **30**(3-4) (1992) 143–150
11. Dyer, F.: When it pays to waggle. *Nature* **419** (2002) 885–886
12. Camazine, S., Sneyd, J.: A model of collective nectar source by honey bees: selforganization through simple rules. *Journal of Theoretical Biology* **149** (1991) 547–571
13. Barth, F.: Insects and flowers: The biology of a partnership. Princeton University Press, Princeton, New Jersey (1982)
14. Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., Wehner, R.: A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems* **30**(1-2) (2000) 39–64
15. Müller, M., Wehner, R.: Path integration in desert ants, *Cataglyphis Fortis*. *Proceedings of the National Academy of Sciences* **85**(14) (1988) 5287–5290
16. Collett, T.S., Graham, P., Durier, V.: Route learning by insects. *Current Opinion in Neurobiology* **13**(6) (2003) 718–725
17. Collett, T., Collett, M.: How do insects represent familiar terrain. *Journal of Physiology* **98** (2004) 259–264
18. Lucic, P., Todorovic, D.: Computing with bees: attacking complex transportation engineering problems. *International Journal on Artificial Intelligence Tools* **12** (2003) 375–394
19. Nakrani, S., Tovey, C.: On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behaviour* **12** (2004) 223–240
20. Chong, C., Low, M.H., Sivakumar, A., Gay, K.: A bee colony optimization algorithm to job shop scheduling. In: In Proceedings of the 2006 Winter Simulation Conference, Monterey, CA USA (2006) 1954–1961
21. Teodorovic, D., Dell’Orco, M.: Bee colony optimization: A cooperative learning approach to complex transportation problems. In: Proceedings of the 16th Mini - EURO Conference and 10th Meeting of EWGT. (2006)
22. Teodorovic, D., Lucic, P., Markovic, G., Orco, M.D.: Bee Colony Optimization: Principles and Applications. In: 8th Seminar on Neural Network Applications in Electrical Engineering (NEUREL 2006). (2006)
23. Viswanathan, G.M., Afanasyev, V., Buldyrev, S.V., Havlin, S., da Luze, M.G.E., Raposo, E.P., Stanley, H.E.: Lévy flights in random searches. *Physica A: Statistical Mechanics and its Applications* **282** (2000) 1–12