Bee Behaviour in Multi-agent Systems
(A Bee Foraging Algorithm)

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

¹ CoMo, Vrije Universiteit Brussel, Belgium
² MICC-IKAT, Universiteit Maastricht, Netherlands
{nlemmens, anowe}@vub.ac.be,
{steven.dejong, k.tuyls}@MICC.unimaas.nl

Abstract. In this paper we present a new, non-pheromone-based algorithm inspired by the behaviour of bees. The algorithm combines both recruitment and navigation strategies. We investigate whether this new algorithm outperforms pheromone-based algorithms, inspired by the behaviour of ants, in the task of foraging. From our experiments, we conclude that (i) the bee-inspired algorithm is significantly more efficient when finding and collecting food, i.e., it uses fewer iterations to complete the task; (ii) the bee-inspired algorithm is more scalable, i.e., it requires less computation time to complete the task, even though in small worlds, the ant-inspired algorithm is faster on a time-per-iteration measure; and finally, (iii) our current bee-inspired algorithm is less adaptive than ant-inspired algorithms.

1 Introduction

In this paper we introduce a new, non-pheromone-based, algorithm inspired by the social behaviour of honeybees. The algorithm consists of two strategies. First, a recruitment strategy which is used to distribute knowledge to other members of the colony. More precisely, by ‘dancing’ inside the hive agents are able to directly communicate distance and direction towards a destination, in analogy to bees ‘dancing’ inside the hive [1]. Second, a navigation strategy which is used to efficiently navigate in an unknown world. For navigation, agents use a strategy named Path Integration (PI). This strategy is based on PI in bees with which they 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 [2,3].

Pheromone-based algorithms are inspired by the behaviour of ants. For an overview, we refer to [4]. 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 autocatalitic process. Short paths will eventually be preferred but it takes a certain amount of time before such pheromone trails emerge.
Although ant and bee foraging strategies differ considerably, both species solve the foraging problem efficiently. In the field of Computer Science, researchers have become inspired by the behaviour of social insects, since the problems these insects cope with are similar to optimization problems humans wish to solve efficiently, for instance, the Shortest Path Problem. Ant-inspired algorithms are already used to address such problems successfully [2]. Bee-inspired algorithms are less extensively studied and research into them only started recently. For instance, [5,6,7,8] all present bee-inspired algorithms which pose solutions to different types of problems by employing bee recruitment behaviour. In [2] the navigation behaviour of bees is investigated and applied in a robot. However, these algorithms use only one aspect of bee behaviour, i.e., the recruitment behaviour or navigation behaviour respectively. As such, there are still two important open issues. First, recruitment and navigation algorithms are currently only studied separately; a combined algorithm is undiscovered land. Second, since a combined bee-inspired algorithm currently does not exist, comparative studies have not yet been performed. We want to investigate whether our bee-inspired algorithm poses a better solution to the foraging problem than an ant-inspired algorithm. More precisely, we want to investigate whether paths emerge faster with our bee-inspired algorithm. Such a comparative study would need to focus on the efficiency, scalability, and adaptability of the algorithms.

Our research addresses both issues. First, a new bee-inspired algorithm, which implements both bee recruitment and bee navigational strategies, is presented. Second, we have developed a simulation environment, named BEEHAVE, in which foraging algorithms can be compared directly. Using BEEHAVE, we are able to compare the bee-inspired algorithm with an ant-inspired algorithm (with features of Ant Colony System and MAX-MIN Ant System [2]). Extensive experiments have been performed with respect to efficiency and scalability. Moreover, we are able to give an indication of the adaptability of the new bee-inspired algorithm. In this paper, we present an overview of our research [9].

The remainder of this paper is structured as follows. In Section 2 we describe the biological background of bee behaviour. Section 3 describes how to model bee behaviour. Section 4 describes the simulation environment and the experiments. Finally, in Section 5 we present the conclusion and two options for future research.

2 Biological Background

Foraging honeybees display two types of behaviour, i.e., recruitment and navigation behaviour. 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 [1]. 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. The angle between the sun and the axis of a bee’s waggle segment on the vertical comb, represents the azimuthal angle between the sun and a target location, i.e., the direction in which a recruit should fly [1,10,11](see Figure 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 understood. 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 an-
gles steered and all distances covered [2]. A PI vector represents the insects 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 com-
putationally simple approximation [3]. Using this approximation, the insect is able to
return to its destination directly. More precisely, when the path is unobstructed, the in-
sent 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 [14][15] to
solve the problem. Obviously, bees are able to fly, i.e., 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. The landmarks divide the entire
path in segments and each landmark has a PI vector associated with it. This behaviour
decreases navigation errors and ensures robustness. In the remainder of this paper, we
refer to a home-pointing PI vector as a Home Vector (HV). PI is used in both exploration
and exploitation. During exploration insects constantly update their HV. It is however,
not used as an exploration strategy. During exploitation, the insects update both their
HV and the PI vector indicating the food source, and use these vectors as a guidance to
a destination.
3 Modelling Bee Behaviour

In contrast to existing algorithms [5,6,7,8], our new algorithm combines both biological behaviours previously mentioned. First, recruitment behaviour is implemented in analogy with biological bees’ dance behavior. Agents 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 from other agents in the hive, or to exploit their own search experience, if available. As mentioned earlier, 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. More precisely, 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. Second, the navigation behaviour used in the bee-inspired algorithm either exploits previous search experience (of the agent itself or of another agent in the hive) or lets the agent explore the world using an exploration strategy similar to a Lévy flight [16]. Exploiting previous search experience is guided by the 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 Ant Colony Optimization [4]. It implements both recruitment and navigation behaviour 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 behaviour is performed. State changes are outlined in Algorithm 1. 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. Previous search experience is represented by a PI vector indicating a food source. If such experience is available, the agent will leave the hive to exploit the previous search experience. Agent state ‘Exploitation’ indicates that the agent is exploiting previous search experience. An agent either exploits its own search experience or acquires a PI vector from other agents inside the hive. 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 toward the hive. The agent’s return path depends on the same HV as with agent state ‘HeadHome’.

1 The last function, 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 this paper, DaemonActions() is not used.
Algorithm 1. Agent internal-state changes

1: if State is StayAtHome then
2: if Vector exists then
3: Exploitation
4: end if
5: else if Agent not AtHome then
6: if Agent has food then
7: CarryingFood
8: else if Depending on chance then
9: HeadHome, Exploration or Exploitation
10: end if
11: else if Exploit preference AND state is AtHome then
12: if Vector exists then
13: Exploitation
14: else
15: Exploration
16: end if
17: else if StayAtHome preference AND state is AtHome then
18: if Vector exists then
19: Exploitation
20: else
21: StayAtHome
22: end if
23: else
24: Exploration
25: end if

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. Our algorithm uses an exact PI vector calculation which rules out the directional and distance errors that biological PI is prone to make [3,15]. It does, however, work in a similar way. A new 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 as:

\[ b = \sqrt{a^2 + c^2 - 2ac \times \cos \beta} \]  

(1)

Using Equation 1, \( a \) represents the distance traveled since the last turn was made, \( c \) the old homing distance, and \( b \) the new homing distance. \( \beta \) is the angle turned with respect to the old homing angle. Using Equation 1 we can now calculate \( \alpha \) (the angle used for adjusting the old homing angle), once again by using the cosine rule.

\[ \alpha = \arccos \left( \frac{a^2 - b^2 - c^2}{-2bc} \right) \]  

(2)

Values obtained by Equation 1 and Equation 2 are used to construct the new PI vector.
Bee behaviour’s main feature is that it naturally constructs a direct, optimal path between a starting point (i.e., the hive) and a destination (i.e., the food source). One could argue that bee behaviour is a natural way of constructing options in a Markov Decision Process (MDP). Options are courses of action within a MDP whose results are state transitions of extended and variable duration [17]. Such courses of action have proven very useful in speeding up learning and planning, ensuring robustness and allowing the integration of prior knowledge into AI systems [18]. An option is specified by a set of states in which the option can be initiated, an internal policy and a termination condition. If the initiation set and the termination condition are specified, traditional reinforcement learning methods can be used to learn the internal policy of the option. In bee behaviour, the basic actions consist out of moving in different directions over the nodes in a MDP (i.e., the foraging world). The option’s policy is represented by the (artificial) bee’s PI vector, where the starting state is the hive and the termination state is the food source location.

4 Simulation Environment and Experiments

To conduct comparative experiments with the bee-inspired algorithm and the ant-inspired algorithm, we created a simulation environment. This environment is called BEEHAVE and is illustrated in Figure 2.

Fig. 2. The BEEHAVE tool

To obtain our data, three experiments have been performed. The experiments were conducted in (i) a small-sized world (i.e., Experiment 1; 110 cells), (ii) a medium-sized world (i.e., Experiment 2; 440 cells), and (iii) a large-sized world (i.e., Experiment 3; 2800 cells). Experiment 1 and 2 each contain five different problem cases (i.e., unobstructed, obstructed, food-source displacement, obstructed with food-source
displacement, and multiple foodsources). Experiment 3 only consists of one problem case, i.e., the unobstructed problem case. Each experiment is executed with both the ant-inspired algorithm and the bee-inspired algorithm (i.e., our new algorithm). Experiment 1 is executed with 50 and 100 agents, while Experiment 2 is executed with 100 and 250 agents. Choosing higher numbers of agents in either of the two experiments leads to agents flooding in the world, preventing any path from arising. The results of Experiment 1 and 2 are used to obtain our main conclusions. Experiment 3 is used to determine how scalable the algorithms are. The algorithms’ scalability is measured with respect to the world size and the number of agents used. In Experiment 3 the number of agents is set to 500.

The comparison is based on efficiency, scalability and adaptability. In Figure 3(a), an example of a medium-sized world is presented. Figure 4 and 5 present the corresponding result figures. The former shows a histogram of the total iterations needed for completing the foraging task at hand. The latter shows a histogram of the average computation time needed per iteration.

Considering efficiency, in Figure 4, we can observe that the bee-inspired algorithm is more efficient, since it uses significantly fewer iterations to complete the task at hand. With an increasing number of agents, the (relative and absolute) efficiency of the algorithm rises. These are typical results found in this research, i.e., they occur in every experiment performed.

In Figure 5(a), we present a histogram of the average computation time needed per iteration in a medium-sized experiment with 100 agents. We observe that the algorithms on average will settle around a computation time of 108ms and 106ms per iteration, respectively. In Figure 5(b) we observe that with 250 agents, the bee-inspired algorithm has a mean of 353ms while the ant-inspired algorithm’s mean is 341ms and has a wide spread. Even though a statistical test reveals that in both cases, the difference is significant in favour of the ant-inspired algorithm, the total computation time required to complete the task is still much lower for the bee-inspired algorithm. Once again, these are typical results; they occur in every small- and medium-sized experiment performed.

Considering scalability, we take into account (i) increasing the number of agents and (ii) increasing the size of the world. With respect to agent scalability, in Table 1
Fig. 4. Histogram of the number of iterations needed in medium-sized, basic-case experiments, with a number of agents as indicated. Black indicates the occurrences for the bee-inspired algorithm. Grey indicates the occurrences for the ant-inspired algorithm. Results are obtained after 300 experimental runs.

Table 1. Performance ratios between the ant-inspired algorithm (Pb) and the bee-inspired algorithm (NPb). I.e., \( \frac{Pb}{NPb} \) ratio. Ratios marked with a (*) are influenced by the maximum number of timesteps available. Due to the fact that experiments are terminated after 2500 timesteps, the ant-inspired algorithm was not always able to complete the task set while the bee-inspired algorithm always did. The marked ratios values therefore could actually be even higher if we allowed for more timesteps.
(a) Using 100 agents.

(b) Using 250 agents.

Fig. 5. Histogram of the average computation time needed per iteration in medium-sized, basic-case experiments, with a number of agents as indicated. Black indicates the occurrences for the bee-inspired algorithm. Grey indicates the occurrences for the ant-inspired algorithm. Results are obtained after 300 experimental runs.

we can observe that when we increase the number of agents and keep the world size constant, ratios decrease (i.e., the ant-inspired algorithm is more scalable with respect to the number of agents). With respect to world scalability, in Table II we can observe that when we increase the world size and keep the number of agents constant, ratios increase (i.e., the bee-inspired algorithm is more scalable with respect to the size of
Overall, the bee-inspired algorithm is more scalable than the ant-inspired algorithm since it finishes its tasks much faster. Considering adaptability, we performed an experiment in which the task set was the Deneubourg Bridge [19]. Figure 6 shows the world in which the experiment is performed. In this two-bridges world, the short path is blocked after a certain number of timesteps. The ant-inspired algorithm performs better than the bee-inspired algorithm in such a world, see Figure 6. This is because of the fact that by using pheromone trails, the ant-inspired algorithm has more information about the world than the bee-inspired algorithm. The latter will try to exploit its most direct path even when this most direct path is blocked. The ant-inspired algorithm however, will eventually move towards the unblocked path due to the accumulated pheromone on this path. To enable the bee-inspired algorithm to obtain this environmental information, some extra features have to be added, such as landmark navigation. The results indicate that the ant-inspired algorithm is more adaptive than the current bee-inspired algorithm.

5 Conclusion

Taking into account the results of the experiments in this research, we may conclude that our bee-inspired algorithm is significantly more efficient than the ant-inspired algorithm when finding and collecting food.

Concerning scalability, we may conclude that the ant-inspired algorithm is most scalable with respect to the number of agents used, while the bee-inspired algorithm is most scalable with respect to the size of the world used. The latter might be a desirable feature; multi-agent systems are mostly applied in large worlds. Furthermore, we may conclude that even in smaller worlds, our bee-inspired algorithm requires less total computation time than an ant-inspired algorithm, even if in some cases, the latter requires less computation time per iteration. Besides these benefits, we have to note that our bee-inspired algorithm is less adaptive than the ant-inspired algorithm.

Currently, we are extending the recruitment behaviour of our artificial bees. More precisely, we are adding quality assessment for the artificial bee’s decision on dance
following. Furthermore, we are extending the simulation environment to make it able to construct worlds that are even more dynamic (i.e., moving obstacles and food sources varying in quality). In order to make the algorithm more adaptive, we are investigating whether it is possible to navigate on landmarks. Such landmarks could possibly be created (and decided on) through cooperation between agents or eventually created centrally by the Deamonactions() function (as known in ACO). We are also evaluating to which problems the algorithm could be applied and whether graph-based problems could also make use of path approximation via PI vectors.

We give two more options for future research. First, it might be interesting to construct a hybrid algorithm. By extending the bee-inspired algorithm with, for example, pheromone direction markers, we could improve the algorithm’s adaptability possibly without decreasing its efficiency or scalability. Second, by combining PI strategies with potential field searching [20], we could improve local search.

References


