Stigmergic Landmark Foraging

Nyree Lemmens
Faculty of Humanities and Sciences, MICC,
Maastricht University
P.O. Box 616, 6200 MD
Maastricht, The Netherlands
n.lemmens@micc.unimaas.nl

Karl Tuyls
Faculty of Industrial Design, Eindhoven
University of Technology
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
k.p.tuyls@tue.nl

ABSTRACT

In this paper, we describe a nature-inspired optimization algorithm based on bee foraging behavior. This algorithm combines the high performance of bee path-integration navigation with ant-like stigmergic behavior in the form of landmarks. More precisely, each individual landmark can be created at any walkable state in the environment and contains a collection of direction markers with which visiting agents can find their way in an unknown environment. A landmark can either be represented by an agent or any other information distributing object (e.g., a RFID). Essentially, we implement ant recruitment behavior based on pheromone. However, instead of using attracting or repelling pheromone in every state of the environment, we only update directional information at key locations in the environment. The resulting algorithm, which we call Stigmergic Landmark Foraging (SLF), proves to be very efficient in terms of building and adapting solutions.

Categories and Subject Descriptors

Swarm Intelligence [**Multi-Agent Systems**]: Landmark navigation

Keywords

Swarm Intelligence, Multi-agent Systems, Landmark navigation

1. INTRODUCTION

In previous work we presented Bee System as an alternative to Ant Colony Optimization (ACO) in foraging domains [17]. Foraging is an interesting problem in the domain of Multi-Agent Systems (MAS). The problem entails that a group of agents has to gather objects in a complicated, potentially dynamical environment. 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 (i.e., recruitment) and will follow this trail towards a destination. This is known as an autocatalitic process. More precisely, the more ants follow a trail, the more that trail becomes attractive for being followed. Short paths will eventually be preferred. Since ants follow pheromone trails towards their destination, it may be clear that the ant's navigation strategy is also guided by pheromone [11].

In contrast, 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 a strategy named Path Integration (PI). 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 [15, 21]. For recruitment, bees employ a direct strategy by dancing in the nest. Their dance communicates distance and direction towards a destination [27].

In previous research, we introduced a foraging algorithm inspired by bees and compared it to an Ant System [17]. Our comparison showed that the bee-inspired, non-pheromone-based algorithm clearly outperformed the ant-inspired, pheromone-based algorithm in relatively unobstructed environments; more precisely, the bee algorithm was able to collect all items present in the environment in about three times less time steps than the ant algorithm [17]. However, we also found that in more constrained and/or dynamical environments, such as the Deneubourg Bridge [10], the bee algorithm tends to get stuck behind obstacles [17]. Unlike the ant algorithm, the bee algorithm is only able to learn from any mistakes at one location (i.e., the hive) and as such is less adaptive.

To account for the adaptability problem we introduced inhibition pheromones, resulting in the algorithm 'Bee System with inhibition Pheromones' (BSP) [16]. More precisely, agents can place pheromones in cells that can be considered undesirable, for instance cells that lead to a dead end. In other words, this algorithm is a hybrid, integrating some features of Ant Systems into a bee-inspired algorithm. The major advantage of this algorithm is that its performance is equal to our initial Bee System algorithm while being at least as adaptive as the ant-inspired algorithm. However, the major downside of this approach is that inhibition pheromones need to be simulated somehow, which is not a trivial issue, especially in embodied systems.

Therefore, in [18], we explored the possibility of using landmarks instead of pheromones. More precisely, we extended our previously introduced Bee System [17] with landmark navigation with which agents can learn landmarks that together function as routes. Each landmark represents a segment in the total route [6] and indicates what follow-up action to take to get to the next segment. Moreover, each agent is able to contribute to the success of a landmark route. The more agents follow a route, the more that route becomes attractive for being followed. Summarizing, [18] presents a stigmergic landmark algorithm. With respect to pheromone-based algorithms, such a solution is easier to realize in the physical world since landmarks are either already physically available, or could easily be placed, in the environment by means of, for example, RFID tags. In comparison to BSP, the algorithm is at least as efficient when applied to static environments. However, due to its higher computation time, the algorithm has longer run

Cite as: Stigmergic Landmark Foraging, Nyree Lemmens, Karl Tuyls, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 497–504

times.

In this paper, we present Stigmergic Landmark Foraging (SLF) which extends the algorithm from [18] so that it can be applied in dynamic environments. The extension addresses the route-loop problem that may emerge in dynamic environments. More precisely, due to landmark decay, a route may be broken. When an agent relinks the route it may be attached to a previous visited landmark which may result in a route loop. Moreover, we test a new recruitment heuristic in order to further improve efficiency. To show the performance of SLF, we compare it to BSP in a set of basic static experiments. Furthermore, to illustrate the performance difference with the new heuristic, we apply SLF to a set of environments with multiple, different quality goals. In such experiments, quality may represent goal priority. A high-priority goal should have a higher preference than a low-priority goal. Finally, we show the algorithm's adaptability capabilities in a dynamic environment and its robustness to goal-quality assessment errors.

The remainder of this paper is structured as follows. First, we present an overview of related work in the area of landmark navigation. Next, we present our new landmark algorithm and show its effectiveness in a set of experiments. Finally, we conclude and look at future research.

2. RELATED WORK

In this paper, we focus on landmark navigation in large groups. More precisely, we focus on swarm communities like bee colonies. Individuals of such colonies are limited in their means to navigate over typically unknown environments due to limitations in, for example, brain capacity. However, where the individual fails, the swarm as a whole is able to perform complex tasks such as finding the shortest route to a food item.

In bee communities, landmark navigation is used as an additional navigation strategy next to Path Integration [7]. It actually serves as a strategy to make PI navigation more accurate [2, 8]. Each member of the colony is able to segment the traveled path and measure its traveled distance by recognizing landmarks and acting on them once they are recognized. The distance that spans between two consecutive landmarks is represented by a so-called local vector. Each local vector is attached to the landmark that is seen at the beginning of the segment, such that when a landmark is recognized, the bee recalls the associated local vector that guides it to the next landmark. Local vector memories can be linked together in a sequence so that, even in the absence of the triggering landmark, memory recall occurs [5, 6, 9, 19]. Each bee furthermore keeps an up-to-date global vector which spans between the start location and the destination [5]. However, when bees move over familiar territory, local vectors activated in a sequential fashion take precedence over the global path integration vector. The global vector thus serves as a backup strategy [4].

Bees basically classify landmarks into two groups, (i) global landmarks and (ii) local landmarks. First, global landmarks are used to guide the bee to the rough area of the goal and can be used over large spatial distances. This panorama (i.e., the global landmarks) determines which actions of movement are recalled [9, 26]. Second, local landmarks are close to the goal (or the hive) and are therefore a natural choice for pinpointing the goals position [3, 9, 26]. However, local landmarks are not as reliable as global landmarks. Local landmarks (e.g., a leaf) tend to change over time whereas global landmarks (e.g., a mountain range) tend to be stable. Research has shown that in absence of known global landmarks, bees are still able to navigate towards the goal albeit not very accurately. Local landmarks seem to be used as a secondary mechanism of navigation. The local landmarks are not attended when a familiar route and known global landmarks are present. Thus, bees preferably attend to global landmarks [26].

In recent years, researchers have started to use landmark navigation behavior to create robots which can navigate in unknown environments autonomously. In [1], a single robot is presented which builds an internal map of the environment in order to navigate through it. Although the robot's control system is designed as a MAS, it is solely used to determine which control system (for example, the pilot system) of the robot get access to the robot. By using a bidding mechanism the most urgent control system gets access. In [12, 13], a single robot is able to navigate through an environment based on image matching. The robot is allowed to take snapshots (i.e., images) while recording positional information. By mathematically comparing two snapshots, the robot is able to calculate an average displacement vector with which it can move towards a goal. Together these vectors make a displacement field with which a robot can move from any point in the environment towards its destination. [20] also demonstrates an image matching approach to landmark navigation and shows its functionality in a robot with analog electronic hardware. [14] presents a single robot that uses the Perception-Action architecture which is a neural computation architecture proposed to solve a wide variety of control problems requiring learning capabilities. By using this architecture, the robot is able to learn how to reach any goal from any other landmark by moving around the goal. [22] presents a robot with a self-organizing mechanism to build a representation of the environment based on landmark recognition. The robot first starts with a map-building effort and once the map is complete the map interpretation starts which consists of map planning and execution. Note that all these researches are based on single robots. Furthermore, most researches are based on offline learning, i.e., a map has to be learnt before it can be used.

3. THE SLF ALGORITHM

In this section we will describe the algorithm developed. First, for the sake of clarity, we give a short overview of the original bee-inspired algorithm from [17]. Next, we describe the novel SLF algorithm. In the following, when we speak of 'the hive' we indicate the starting position of the agent. 'The goal' indicates a destination with a collection of items present.

3.1 The original bee-inspired algorithm

The bee algorithm implements both recruitment and navigation behavior. *Recruitment behavior* is implemented in analogy with biological bees' dance behavior [16, 17, 23]. Agents (artificial bees) share information on previous search experience (i.e., the direction and distance toward a certain goal) 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 goals at a shorter distance from the hive than the goals 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 called a Lévy flight [25]. 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. In the remainder of this paper we refer to a goal-directed PI vector as a Goal Vector (GV), a hive-directed PI vector is referred

to as a Homing Vector (HV).

The general structure of our bee-inspired algorithm is quite similar to that of algorithms in ACO [11]. It implements both recruitment and navigation behavior and consists of three functions.

(i) $ManageBeesActivity()$ 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.

First, agent state '*Explorer*' indicates the agent is exploring the environment for goals and performing Lévy flights to optimally cover the search space [24]. During exploration, the agent continuously updates its HV.

Second, agent state '*Carrier*' indicates the agent found a goal and is returning an item from the goal to the hive. The actions an agent takes during the return are guided by the HV. During the return, the agent continuously updates its HV in order to have an accurate vector of guidance. The GV is set to the $HV + 180°$ the moment the agent arrives at the goal.

Third, '*Dancer*' indicates that an agent arrived at the hive with directional knowledge of a goal (i.e., the GV). In order to recruit other colony members for this goal, the agent starts a virtual dance which other colony members can observe. The dance communicates the directional information directly to the observers.

Fourth, agent state '*Rester*' indicates the agent remains in the hive and is resting. It either did not find a goal by exploration or it did find a goal and recently stopped dancing for it.

Fifth, agent state '*Observer*' indicates an agent is looking for dancers to observe. Whenever it finds one, it determines whether the danced 'advertisement' is of high enough quality to adopt.

Sixth, agent state '*Recruit*' indicates an agent adopted the directional information from a dancer and is traveling to the goal. Its actions are guided by the GV which is continuously updated during the travel to the goal. During exploitation phase, the HV is also continuously updated.

(ii) $CalculateVectors()$ is used to compute the PI vectors for each agent, i.e., the HV and possibly GV indicating the goal. A PI vector essentially consists of two values, one indicating the direction and the other indicating the distance (i.e., the magnitude). A PI vector is always calculated with respect to the previous one. In order to calculate the new distance, we use the cosine rule and rewrite it to Equation 1a.

$$b = \sqrt{a^2 + c^2 - 2ac \times cos\beta} \tag{1a}$$

$$\alpha = \arccos\left(\frac{a^2 - b^2 - c^2}{-2bc}\right) \tag{1b}$$

In Equation 1a, a represents the distance traveled 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 angle. Using Equation 1a we can now calculate α (i.e., the angle used for adjusting the old angle), once again by rewriting the cosine rule, see Equation 1b. Values obtained by Equation 1a and Equation 1b are used to construct the new PI vector.

(iii) $DaemonActions()$ 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 the original bee-inspired algorithm, $DaemonActions()$ is not used.

3.2 The SLF algorithm

In order to improve the original bee-inspired algorithm, we extend it by adding landmark navigation. Essentially, this means that agents are able to create or represent sub-hives in the environment. In this research, agents create sub-hives in the environment and do not represent the landmarks by themselves. Each sub-hive contains directional information, in the form of HV's and/or GV's, to other sub-hives (or the goal) and form together a network of directed vectors from hive to goal and/or goal to hive. Figure 1 shows a basic example of such a sub-hive network. Because of the resemblance to biological landmarks, in the remainder of this paper we refer to these sub-hives as landmarks. Each landmark is essentially a state (e.g. grid block, node) in which it is profitable to have additional (heuristic) information in order to guide the search to a better solution.

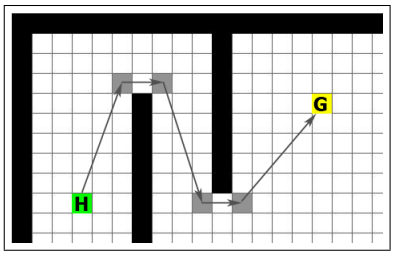


Figure 1: Basic example of a landmark network. 'H' and 'G' respectively represent the hive and goal. The grey squares represent the landmarks. The arrows represent a landmark route.

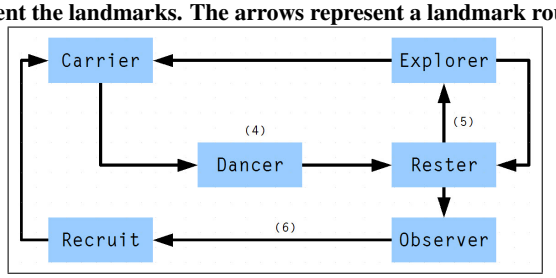


Figure 2: Flow between each of the internal states. The numbers which can be found at various edges represent the equations numbered equally in the paper.

With respect to the original bee-inspired algorithm each agent's memory is slightly increased with three fields.

First, the temporary vector (TV) represents the vector from the last known landmark (or hive) to the current landmark (or goal). On arrival at a landmark (or goal), the TV is reset. The TV is essentially the same as a local vector in biological landmark navigation [5].

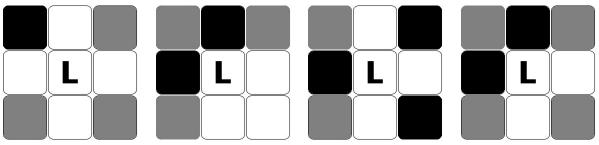
Second, an agent is able to remember which route it is actually following. Each landmark can thus represent multiple routes towards the goal (and/or back to the hive).

Third, an agent is able to remember the quality of a found goal.

The general algorithm structure is based on the original bee-inspired algorithm and thus consists of three functions, i.e., (i) $ManageBeeBehavior()$, (ii) $CalculateVectors()$, and (iii) $DeamonActions()$. With respect to these functions the first and last one have a different implementation than in the original bee-inspired algorithm.

a) The behavior of agents in $ManageBeeBehavior()$ is determined by the six possible internal states and is inspired by agent behavior as described in [23]. In Figure 2, we show the flow between each of the internal states.

First, agent state '*Explorer*' indicates the agent is exploring the environment for goals and performing Lévy flights to optimally cover the search space [24]. During exploration, the agent continuously updates its HV. While covering the search space, the agent is able to detect key locations (i.e., landmarks) in the environment which are then used to store the agents' TV. The landmark detection conditions can be found in Figures 3(a), 3(b), 3(c) and 3(d). Essentially, a key location is found on a corner of an obstacle, an open corner,



(a) Obstacle (b) Open (c) T-corridor (d) Corridor

Figure 3: Situations in which a key location (i.e., landmark) is detected. 'L' represents the landmark location. Grey squares indicate possible obstacles, black squares indicate necessary obstacles. Constellations may be rotated and/or mirrored. In none of the cases a hive or goal may be present.

SL [2]	ALR	Explorer Consequence
False	False	Create new landmark with new agent landmark route
False	True	Create new landmark with known agent landmark route
True	False	*Not possible*
True	True	IF LHV does not exist: continue exploration without creating new landmarks, ELSE IF LGV exists: probability to become recruit.

Table 1: Explorer Landmark creation conditions. SL = Is State a Landmark?, ALR = Is Agent Landmark Route Known?

a corridor in a T-shape, and in a corner of a corridor. These conditions are sufficient for contructing an efficient solution to the foraging problem. Connections between such landmarks ensure an efficient path around obstacles. Note however that landmark detection conditions may differ according to the problem domain. For example, in network routing a landmark may be detected if a node has a certain queue, bandwidth, and/or congestion. During exploration, the TV is stored in the landmark as a Landmark Homing Vector (LHV), indicating the next location to move to in order to arrive home. Moreover, each LHV is linked to a particular route. Each explorer is thus able to create a distinct route towards its home.[1] If an agent arrives in a landmark, it first checks whether the route it is on, is already known inside the landmark. In Table 1 this is indicated in the field ALR. If an agent's TV is obstructed within the agents view range, it is not allowed to create any more landmarks during its exploration phase. Explorer Landmark creation conditions can be found in Table 1. Obviously, these conditions are only valid if the TV is not directly (i.e., within the agent's view range) obstructed.

Second, agent state '*Carrier*' indicates the agent found a goal and is returning an item from the goal to the hive. As a carrier arrives at a goal it determines the quality of the site according to Equation 2. Let the quality assessment Q of agent i for goal j be

$$Q_{ij} = GQ_j + \epsilon_n^i, \tag{2}$$

where $GQ_j \in [0, 1]$ is the quality of goal j (set by the user) and ϵ_n^i the goal-quality-assessment error of agent i due to noise. Goal quality may represent for example goal priority. The actions an agent takes during the return are guided by the LHV of the landmarks the agent visits while following a particular route back home. Each visited landmark is also used to create/reinforce the Landmark Goal Vector (LGV). On arrival at a landmark, the agent stores its TV in the LGV and then resets the TV. If a carrier is not able to follow its route back (i.e., the LHV is blocked or the indicated follow-up landmark is non-existent) it starts exploring for landmarks. If it finds one it tries to adopt another landmark-route home. The carrier is not allowed to create any LGV's if it altered its route. Carrier landmark creation conditions can be found in Table 2. To prevent excessive carrier exploration efforts, a carrier agent has a certain

[1]Explorers are not allowed to extend existing routes. Extending by explorers may result in route looping due to landmark decay.

SL [2]	ALR	Explorer Consequence
False	False	Create new landmark with new agent landmark route
False	True	Create new landmark with known agent landmark route
True	False	*Not possible*
True	True	IF LGV does not exist: set LGV with known agent route, ELSE IF Agent Goal Quality > Landmark Goal Quality then overwrite LGV

Table 2: Carrier Landmark creation conditions. SL = Is State a Landmark?, ALR = Is Agent Landmark Route Known?

lifespan (in time steps). In this research, carriers are sure to stay alive for the time it takes to, at least, travel back and forth along the longest route $((h \times w)^2$ time steps, where h is the height of the environment and w is the width of the environment in cells). After this period, the agent has an increasing probability of death in which case it looses its resource and is 'reborn' inside the hive. The probability of death increases linearly with the number of visited cells after its minimal lifespan.

Third, '*Dancer*' indicates that an agent arrived at the hive with directional knowledge of a landmark or possibly the goal directly. This knowledge is represented in the TV and the landmark route the agent followed. In order to recruit other colony members for its goal, the agent starts a virtual dance which other colony members can observe. The dance communicates the directional information and route directly to the observers. High-quality goals are preferred over low-quality goals. Moreover, short-distanced goals are preferred over long-distanced goals. To normalize a dancer's dance, we use a Quality Cumulative (QC) which sums all distinct goal qualities, see Equation 3.

$$QC = \sum_{i=1}^{B_d} Q_{ij} \tag{3}$$

In Equation 3, B_d represents the number of dancers in the hive and $\sum_{i=1}^{B_d} Q_{ij}$ only counts distinct qualities. Duplicate qualities are discarded. The resulting dance strength is determined according to Equation 4,

$$DS_i = \begin{cases} max\{\gamma(\frac{Q_{ij}}{QC+Q_{ij}}), 0\} & : \quad init. \\ DS_i - \epsilon_d & : \quad Q_{ij} > \epsilon_t \\ 0 & : \quad Q_{ij} \leq \epsilon_t \end{cases}, \tag{4}$$

where DS_i indicates the dance strength for dancer i, Q_{ij} is a stochastic value for the quality of goal j as observed by agent i. QC represents a Quality Cumulative which scales the observed goal quality and, in this research, indicates the total quality of all goals that are danced for in the hive, see Equation 3. $\gamma > 0$ is a parameter which represents the proportionality between the goal quality and the initial dance strength. For example, setting $\gamma = 300$ will result in a dance strength of 300 for a goal with quality 1. ϵ_d sets a value for the dance strength decay. Extending the previous example, setting $\epsilon_d = 15$ would result in dancing for the specific goal for $300/15 = 20$ time steps. ϵ_t sets a threshold on the goal quality before there will be any dancing for the goal. At the end of the virtual dance, a dancer has to decide whether it rests or recruits itself for its own goal. The self-recruiting probability (i.e., P_r) can be set by the user.

Fourth, agent state '*Rester*' indicates the agent remains in the hive and is resting. It either did not find a goal by exploration or it did find a goal and recently stopped dancing for it. A resting agent either becomes an observer or an explorer. The probability to become an explorer is given by Equation 5,

[2]A state is a landmark whenever it is detected as a possible landmark AND it has a agent route linked to it.

$$P_e = e^{\left(-\frac{1}{2}\frac{{DS_t}^2}{\sigma^2}\right)}, \quad (5)$$

where DS_t indicates the total dance strength of all dancing agents, and σ is a scaling parameter and essentially determines the tendency of an agent to explore. Notice that if $DS_t = 0$, there is no dancing so that $P_e = 1$ and all the observer bees will explore (e.g., initially $DS_t = 0$ so all observer bees will choose to explore). If DS_t is low, the observer bees are less likely to find a dancer and thus will not likely get recruited to a goal. They will, in a sense, be 'recruited to explore' by the lack of the presence of any dance. As DS_t increases, the agents become less likely to explore and, as discussed below, will be more likely to find a dancer and get recruited to a goal. In the latter case the agents look for a virtual dance and determine whether the danced 'advertisement' is of high enough quality to adopt, see Equation 6,

$$P_r = \frac{DS_i}{\sum_{i=1}^{B_d} DS_i}, \quad (6)$$

where DS_i represents the dance strength by agent i, B_d indicates the total number of dancing agents. This way, agents that dance 'stronger' recruit more agents for their goal. Thus, high quality goals are going to be preferred.

Fifth, agent state '*Observer*' indicates an agent is looking for dancers to observe. Before the agents starts to observe, it first determines whether it is (still) profitable to observe. This behavior is also captured in Equation 5.

Sixth, agent state '*Recruit*' indicates an agent adopted the directional information from a dancer and is traveling to the goal using the landmarks' LGV on its route. The recruit essentially validates each LHV and LGV. More precisely, if a recruit is able to arrive at an indicated landmark then obviously the LGV is still usable. Likewise, the agent can compare its TV with the LHV. If the vectors are equal then obviously, the route back home is also still valid. On validation, the recruit reinforces the segments of the routes. Eventually, strong routes will emerge and these routes will prevail over the weak routes.

b) SLF is able to handle dynamic environments. To accomplish this, established landmarks somehow have to disappear if they are not used anymore. Therefore in this research $DaemonActions()$ is used and takes care of the landmark decay. Each landmark's dance strength decays with a factor ϵ_d at each time step.

4. EXPERIMENTAL SET-UP

In this section, we will discuss our experimental set-up, i.e., (i) the type of measurements, (ii) the environments to experiment on and their characteristics, and (iii) the settings of the simulator.

First, in this research we would like to discover how SLF performs with respect to (a) efficiency, (b) scalability, and (c) adaptability. These three characteristics are most important in MAS'. A MAS typically has to deal with a large number of agents and it is used to solve (increasingly) large and complex problems which may be dynamic of nature [11]. *Efficiency* relates to the learning performance (i.e., knowledge acquisition and knowledge usage). If a MAS is efficient it will solve its task as fast as possible while using as few resources as possible. In this research, we measure efficiency by the average number of time steps the algorithm uses to complete the task. Moreover, by measuring the average computation time per time step, we also monitor time-related resources. *Scalability* is the ability to cope with an increasing problem size and/or decreasing resource availability. A MAS' scalability is thus partly related to its efficiency. Obviously, an efficient MAS is more scalable than an inefficient MAS. To measure scalability in this research, we perform the experiments with different (increasing) colony sizes in the MAS. Moreover, we experiment on two different-sized environments. In these experiments, a MAS that shows a small decrease in efficiency in an increasingly complex problem is more scalable than a MAS that shows a large decrease in efficiency. *Adaptability* is the ability to adapt learned behavior. An adaptive MAS is able to learn alternative behaviors when the initial learned behavior is not profitable anymore. Learned behavior may become unprofitable due to environmental changes and/or quality assessment errors by agents. Thus, adaptability relates to robustness. Furthermore, relearned behavior should also be efficient. If a MAS in a dynamic environment is inefficient it apparently learned mediocre or bad performing behavior and as a consequence is not adaptive. In this research, adaptability is measured according to Equation 7,

$$A = \frac{\sum_{i=1}^{m}(1 - \frac{(t_n^m - t_c^m)}{t_t})}{m} \quad (7)$$

where $A \epsilon [0, 1]$ is a measure for average adaptability, m is the number of environmental changes, t_n^m is the time step in which a new strongest route surpasses the strongest route from before the environmental change m, t_c^m is the time step of the environmental change m, and t_t is the total number of time steps. A high value for A indicates high average adaptability, a low value indicates low average adaptability. Route strength is determined according to Equation 8,

$$R_s^j = B_r^j + B_c^j \quad (8)$$

where R_s^j indicates the route strength of route j, B_r^j represents the number of recruits for route j, and B_c^j represents the number of carriers for route j. The highest R_s^j represents the strongest route.

Summarizing, we keep a count on the number of recruits and carriers for each route. The route with the highest count is the strongest route. Whenever an environmental change is performed, we store its corresponding time step (i.e., t_c^m) and the current strongest route. Finally, we store the time step at which a different route surpasses the strongest route from before the environmental change. To test adaptability, we experiment with dynamic environments in which obstacles block learned foraging routes towards a goal. We assume that the strongest route is blocked at an environmental change. Moreover, agents may not have perfect knowledge and can make quality assessment errors. An efficient and scalable MAS applied in a dynamic environment in which agent quality assessments are not perfect is clearly adaptive.

Second, in order to perform the measurements, we apply SLF in a number of problem environments. Each environment consists of a number of square cells. Each cell has maximally four neighbors and may contain a hive, goal, or obstacle. In the latter case, agents are not able to traverse these cells. Each environment contains one hive (i.e., start location) and one goal (i.e., a collection of items). Each agent's assignment is to create a route from the hive to the goal, collect a goal-item and return it to the hive. We construct three sets of experiments, (i) BSP Comparison Set (Figure 4), (ii) Heuristic Comparison Set (Figure 5), and (iii) Robustness Comparison Set (Figure 6). We deliberately use relatively simple (yet increasingly complex) environments which may benefit the understanding of the approach and the results.

EXPERIMENT SET 1: BSP Comparison. In order to get an indication about the performance of SLF we compare its performance to the performance of BSP [16] when applied to the same experiment set. The set consists of two experiments. Experiment 1 and 2 are basic experiments without any obstacles between the hive and the goal. They consists of 400 (20x20) and 900 (30x30)

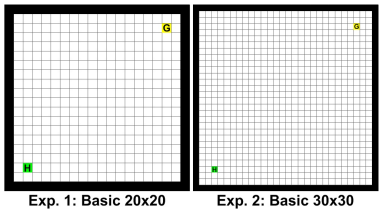


Figure 4: BSP Comparison Set. 'H' and 'G' represent the hive and goal respectively.

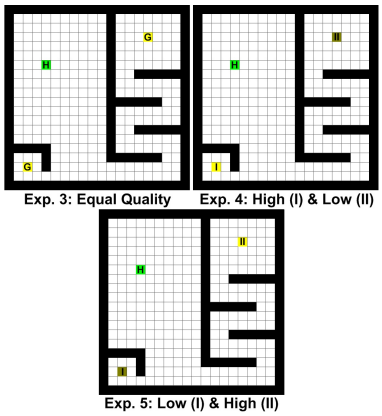


Figure 5: Heuristic Comparison Set. 'H' represents the hive. 'I' always represents the short-distance goal. 'II' always represents the long-distance goal. A dark-colored goal indicates low quality whereas a bright-colored goal indicates high quality.

cells respectively to test for scalability with respect to an increasing problem size. Both experiments contain a hive in the lower-left corner and a goal in the upper-right corner. Obviously, these experiments can be done in all alternative rotations and with different 'constellations' of hive and goal. To test for agent scalability, we perform these experiments with an increasing colony size (i.e., 25 and 50 agents). Each goal contains 300 items.

EXPERIMENT SET 2: Heuristic Comparison. In order to test the recruitment heuristic, Set 2 consists of 3 experiments in which each environment has two paths towards two goals. We test two heuristics. First, a basic heuristic based on goal quality [23]. Essentially, recruitment is being guided only by the goal quality. Indirectly goal quality also represents the distance between hive and goal of the route, i.e., a short-distance goal will receive more virtual dancing than a long-distance goal due to the fact that dancing for short-distance goals will occur more frequently. Thus the recruitment probability will be higher for short-distance goals. However, such a heuristic may not be a guarantee that the most profitable route is used. A high quality, long-distance goal may be more profitable than a low-quality, short-distance goal. Second, to ensure such profitable paths, we include a QC in the heuristic. Recruitment will now be guided by a 'scaled' goal quality. E.g., a low-quality goal 'advertised' among 'advertised' high-quality goals will have an even lower recruitment probability than with the standard heuristic. Eventually, the most profitable path will be preferred. To show effectiveness of each heuristic, we vary the quality of goals and the distance of each goal to the hive. In Experiment 4, the quality of the short-distance goal and the long-distance goal are equal. In Experiment 5, there is a high-quality, short-distance goal and a low-quality, long-distance goal. Experiment 6 presents these goals in the opposite order. Both low-quality and high-quality goals consist of 600 items to clearly show convergence to one of the two paths.

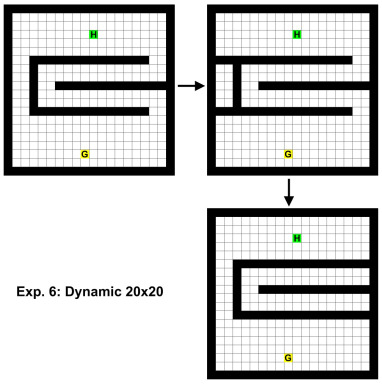


Figure 6: Robustness Comparison Set. 'H' and 'G' represent the hive and goal respectively.

EXPERIMENT SET 3: Adaptability Comparison. As mentioned earlier, adaptability is related to robustness. SLF should be robust (and thus adaptive) to environmental changes and goal-quality-assessment errors. Experiment 6 consists of three cases which together create a dynamic environment. Case 1 shows an environment in which there is a short and a long path to the goal. Moreover, it shows an obstacle construction which the agents have to learn to travel around. Case 2 blocks one path (i.e., the short path). Case 3 opens up one path (i.e., the short path) and blocks the other (i.e., the long path). Thus, agents first need to learn a route from hive to goal during Case 1. At any moment in time, their environment changes (into Case 2) and they have to relearn how to cope with this environment, i.e., forget the short route and relearn/reinforce the long route. Finally, case 3 represent again an environment with the unblocked short route which has to be relearned. This experiment shows the ability of the algorithm to reroute (i.e., 'relearn') a route. We perform Experiment 6 twice. First, agents have perfect goal-quality assessment. Second, agents make goal-quality-assessment errors which result in false assessment of goal quality. Despite of this handicap, the agents should still be able to learn the shortest route towards the goal and gather the goal-items efficiently. A goal contains 300 items.

Third, we use different settings for the three experiment sets.
a) In Set 1, due to BSP being simulated in a different simulator which does not support all extensive measurements, we use the following settings. (i) every agent that starts dancing for a goal stays committed to its goal and recruits itself (i.e., $P_r = 1$), (ii) the experiments are run with a colony size of 25 and 50 agents, (iii) no goal quality assessment errors are allowed (i.e., $\epsilon_n^i = 0$).
b) In Set 2, we only experiment with SLF and set the probability to let a dancer recruit oneself to 0.50 (i.e., $P_r = 0.50$). In experiments without the SQ, we set $QC + Q_{ij} = 1$.
c) In Set 3, we test for adaptability and set the dancer's self-recruitment probability to 0.50 (i.e., $P_r = 0.50$). Note that no adaptability means there is no dancer self-recruitment (i.e., $P_r = 0$) and the initial dance strength is zero (i.e., $\gamma = 0$) and consequently adaptability will be zero (i.e., $A = 0$).

Furthermore, in each of the experiments we use the following settings. (i) There is no quality threshold (i.e., $\epsilon_t = 0$), (ii) dance strength decay is set to 15 (i.e., $\epsilon_d = 15$), (iii) initial dance strength is set to 300 (i.e., $\gamma = 300$) so that the maximal dance has a duration of $300/15 = 20$ time steps, (iv) probability of carrier death is set to 0.001 (i.e., $P_d = 0.001$), (v) the maximal levy flight is set to 15 steps (i.e., cells to travel over), and (vi) the tendency to explore is set a low value (i.e., $\sigma = 10$). The latter setting is determined empirically. Setting a high value for the exploration tendency re-

Set 1 (Efficiency)

Algorithm (# agents)	Experiment 1		Experiment 2	
	μ	σ	μ	σ
BSP (25)	2173.8	14.5	5625.6	52.9
SLF (25)	1228.6	66.9	2028.2	152.2
BSP (50)	1160.6	4.8	2600.6	15.4
SLF (50)	725.9	36.2	1227.6	65.8

Set 1 (Scalability)

Algorithm (# agents)	Experiment 1		Experiment 2	
	μ	σ	μ	σ
BSP (25)	4.76	0.0	4.49	0.0
SLF (25)	18.3	12.2	27.0	17.0
BSP (50)	9.15	0.0	9.05	0.0
SLF (50)	44.1	20.5	64.3	32.6

Table 3: The efficiency (top) and scalability (bottom) of the two algorithms under study in Set 1. Efficiency is measured in time steps. Additionally, we indicate the exploration/exploitation ratio for SLF. Scalability is measured in computation time per time step. e_r indicates the exploration/exploitation ratio. Each experiment contains a goal consisting of 300 items.

sults in degraded efficiency performance due to the low probability of recruitment. To compensate for the inherent randomness in the algorithms, we perform each experiment in every set 30 times. To limit run-time, we set a maximal time-step limit of 12000 time steps per experiment.

5. RESULTS

To have an indication of performance, we compare SLF's performance with BSP's performance [16] when applied to the same experiment set, i.e., Set 1. We compare the algorithms' performance with respect to their efficiency and scalability. Efficiency can be measured by determining the number of time steps the algorithms require to gather all the items present in the goal. Scalability relates to the computation time required to perform this task. As can be observed in the top section of Table 3, SLF outperforms BSP with respect to efficiency. On average, SLF is 2 times faster in solving the problem set. The time-step ratios with both increasing colony size and increasing world size approximately stay equal. For example, the time-step ratio for SLF with increasing colony size in the two different-sized experiments are $725.9/1228.6 = 0.59$ and $1227.6/2028.2 = 0.61$ respectively. Thus, considering the efficiency, both algorithms are equally scalable with increasing colony size and world size. However, SLF needs more computation time per time step as can be observed in the bottom section of Table 3. On average SLF uses 6.9 times more computation time per time step which in the end results in longer run times for SLF. The computation-time ratios stay approximately equal with increasing agent numbers. But SLF's computation-time ratio becomes worse with increasing world size while BSP's computation-time ratio remains approximately equal. This is due to the nature of the two algorithms. BSP uses a binary inhibition pheromone. Either the pheromone is there or it is not. In contrast, SLF needs to perform some 'administration' efforts with respect to landmark strength. The larger the world, the more possible landmarks, the more 'administration' to be done. Obviously, SLF limits the amount of 'administration' by selecting only key locations for landmarks. With respect to an ACO algorithm [16], SLF may thus have a clear advantage since an ACO algorithm typically needs to perform 'administration' for significantly more locations.

Set 2 presents experiments with multiple, different-quality goals. The set is used to explore whether an extended heuristic (i.e., with Quality Scaling) improves performance of SLF. The results, which

Set 2 (Efficiency & Scalability)

Algorithm	Experiment 3		Experiment 4		Experiment 5	
	μ	σ	μ	σ	μ	σ
Efficiency SLF	2993.3	311.0	4246.9	360.3	3737.0	272.8
Efficiency SLF (SQ)	2956.1	442.8	3196.7	216.8	3242.7	238.4
Scalability SLF	42.8	20.3	76.5	37.8	64.6	29.3
Scalability SLF (SQ)	54.6	25.0	59.4	24.6	58.7	23.2

Set 2 (Heuristic)

Algorithm	Experiment 3		Experiment 4		Experiment 5	
	$F1$ (μ)	$F2$ (μ)	$F1$ (μ)	$F2$ (μ)	$F1$ (μ)	$F2$ (μ)
SLF	I (1300.9)	II (2869.5)	I (1085.5)	II (4179.5)	II (1443.5)	I (3657.9)
SLF (SQ)	I (1579.6)	II (2792.8)	I (1066.9)	II (3126.5)	II (1668.2)	I (3191.9)

Table 4: The efficiency (top), scalability (top), and heuristic (bottom) of the two algorithms under study in Set 2. Efficiency is measured in time steps. Scalability is measured in computation time per time step. $F1$ and $F2$ represent the first and second depleted goals respectively and are named as indicated in Figure 5. μ indicates the average time step at which a given goal is depleted. Each goal consists of 600 items.

Set 3 (Efficiency & Scalability)

Algorithm	Experiment 6a		Experiment 6b	
	μ	σ	μ	σ
Efficiency SLF	4221.1	1101.3	4703.9	763.4
Scalability SLF	59.5	30.5	63.0	31.9

Set 3 (Adaptability)

Algorithm	Experiment 6a	Experiment 6b
	$Adaptability$	$Adaptability$
SLF	94%	93%

Table 5: The efficiency (top), scalability (top), and adaptability (bottom) of SLF in Set 3. Efficiency is measured in time steps. Scalability is measured in computation time per time step.

can be observed in Table 4, indicate that both heuristics perform equally well in an experiment with equal quality goals and there is no significant difference in performance (i.e., double-sided t-test with 99% confidence; $t = 0.3766$ for efficiency, $t = -2.0069$ for scalability). With respect to finding the best goal first, we cannot observe any difference either (see bottom section of Table 4. However, SLF with Quality Scaling shows a higher overall efficiency in experiments with different quality goals. Thus, the extended heuristic shows an advantage over the standard heuristic. Quality Scaling scales a goal quality up when it's the only goal being 'advertised' in the hive. Such a scaled goal quality ensures more dancing for the goal and as a consequence more and prolonging recruitment which in the end results in faster item return.

Table 5 shows the results for the dynamic experiments, i.e., Experiment 6a and 6b. The former allows no agent quality assessment errors whereas the latter does allow such errors. The results indicate how robust and adaptive SLF is with respect to dynamics and errors. From the results we can observe that SLF is robust to observation errors. Efficiency performance is only slightly lower. A double-sided T-test with 99% confidence interval reveals the difference is not significant ($t = -1.9735$). Moreover, the adaptability measure, as seen in the bottom section, indicates that adaptability is robust to observation errors since the adaptability percentage only decreases with 1%. Adaptability shows the ability of the algorithm to adapt learned behavior to improve performance.

6. CONCLUSION AND FUTURE WORK

In this research, we presented a novel distributed landmark navigation algorithm for a MAS. More precisely, the algorithm con-

structs stigmergic landmark networks over which agents can travel from and to a goal. In contrast to existing algorithms, we present an online learning approach in which found landmarks can immediately be used. Each landmark represents a segment of the total route, ensuring robustness and accuracy by decreasing inherent accumulating PI errors.

Observing our results, we may conclude that SLF is significantly more efficient than BSP. However, its computation time per time step is approximately 7 times larger which in the end results in longer run times for the experiments. Although SLF performs worse with respect to computation time, it is worth noting that this may not be a significant problem. For example, if SLF would be applied in swarming robots, each robot itself may use significantly more time than 100 ms to move from one location to another. Moreover, in a physical swarming environment, the task is typically divided over multiple independent components. This in contrast to the current simulation in which the task was done by only one processor. With respect to scalability we may conclude that both algorithms are equally scalable. Furthermore, we may conclude that extending the algorithm's heuristic is advantageous. By doing so, the efficiency of the algorithm increases. Finally, we may conclude that SLF is robust to quality assessment errors. If the agents's quality assessment has errors, the performance of the algorithm does not suffer from this.

In future, we would like to apply an algorithm as described in this research to the problem domain of network routing. Foraging and routing have a number of properties in common, such as, (i) finding optimal routes, (ii) constructing robust routes, and (iii) load balancing. It might be interesting to see how an algorithm such as SLF would perform in comparison to state-of-the-art (swarming) routing algorithms such as Dynamic Source Routing (DSR), Ad-hoc On-demand Distance Vector Routing (AODV), and AntHocNet.

7. REFERENCES

[1] D. Busquets, C. Sierra, and R. L. de M'antaras. A multiagent approach to qualitative landmark-based navigation. *Autonomous Robots*, 15:129–154, 2003.

[2] B. Cartwright and T. Collett. Landmark learning in bees. *Journal of Comparative Physiology A*, 151:521–543, 1983.

[3] K. Cheng, T. Collett, A. Pickhard, and R. Wehner. The use of visual landmarks by honeybees: bees weight landmarks according to their distance from the goal. *Journal of Comparative Physiology A*, 161:469–475, 1987.

[4] L. Chittka and J. Kunze. The significance of landmarks for path integration in homing honeybee foragers. *Naturwissenschaften*, 82:341–343, 1995.

[5] M. Collett, T. Collett, S. Bisch, and R. Wehner. Local and global vectors in desert ant navigation. *Nature*, 394:269–272, 1998.

[6] M. Collett, D. Harland, and T. S. Collett. The use of landmarks and panoramic context in the performance of local vectors by navigating honeybees. *Journal of Experimental Biology*, 205:807–814, 2002.

[7] T. Collett. Insect navigation en route to the goal: multiple strategies for the use of landmarks. *Journal of Experimental Biology*, 199:227–235, 1996.

[8] T. Collett, E. Dillmann, A. Ciger, and R. Wehner. Visual landmarks and route following in desert ants. *Journal of Comparative Physiology A*, 170:435–442, 1992.

[9] T. S. Collett and M. Collett. Memory use in insect visual navigation. *Nature*, 3:542–552, 2002.

[10] J. Deneubourg, S. Aron, S. Goss, and J. Pasteels. The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.

[11] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[12] M. O. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff. Learning view graphs for robot navigation. *Autonomous Robots*, 5:111–125, 1998.

[13] M. O. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff. Where did i take that snapshot? scene-based homing by image matching. *Biological Cybernetics*, 79:191–202, 1998.

[14] P. Gaussier, C. Joulain, J. Banquet, S. Leprêtre, and A.Revel. The visual homing problem: An example of robotics/biology cross fertilization. *Robotics and Autonomous Systems*, 30:155–180, 2000.

[15] D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30(1-2):39–64, 2000.

[16] N. Lemmens, S. de Jong, K. Tuyls, and A. Nowé. Bee system with inhibition pheromones. In *European Conference on Complex Systems (ECCS)*, 2007.

[17] N. Lemmens, S. de Jong, K. Tuyls, and A. Nowé. Bee behaviour in multi-agent systems: a bee foraging algorithm. *Lecture Notes in Artificial Intelligence: Adaptive Agents and MAS III*, LNAI 4865:145–156, 2008.

[18] N. Lemmens and K. Tuyls. Stigmergic landmarks lead the way. In *The 20th Belgian-Dutch Conference on Artificial Intelligence (BNAIC)*, 2008.

[19] R. Menzel, K. Geiger, L. Chittka, J. Joerges, J. Kunze, and U. Müller. The knowledge base of bee navigation. *Journal of Experimental Biology*, 199:141–146, 1996.

[20] R. Möller. Insect visual homing strategies in a robot with analog processing. *Biological Cybernetics*, 83:231–243, 2000.

[21] M. Müller and R. Wehner. Path integration in desert ants, *Cataglyphis Fortis*. *Proceedings of the National Academy of Sciences*, 85(14):5287–5290, 1988.

[22] C. Owen and U. Nehmzow. Landmark-based navigation for a mobile robot. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (Complex Adaptive Systems)*, 1998.

[23] K. M. Passino and T. D. Seeley. Modeling and analysis of nest-site selection by honeybee swarms: the speed and accuracy trade-off. *Behavioral Ecology and Sociobiology*, 59:427–442, 2006.

[24] M. van Dartel, E. Postma, and H. van den Herik. Universal properties of adaptive behaviour. In H. Blockeel and M. Denecker, editors, *Proceedings of the 14th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'02), Leuven, Belgium*, pages 59–66, 2002.

[25] G. M. Viswanathan, V. Afanasyevc, S. V. Buldyrev, S. Havlin, M. G. E. da Luze, E. P. Raposof, and H. E. Stanley. Lévy flights in random searches. *Physica A: Statistical Mechanics and its Applications*, 282:1–12, 2000.

[26] A. N. Vlasak. Global and local spatial landmarks: their role during foraging by columbian ground squirrels (*Spermophilus columbianus*. *Animal Cognition*, 9:71–80, 2006.

[27] K. von Frisch. *The dance language and orientation of bees*. Harvard University Press, Cambridge, Massachusetts, 1967.