



Multi-agent Physical A* with Large Pheromones

ARIEL FELNER

felner@bgu.ac.il

Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, 84104, Israel

YARON SHOSHANI

shoshai@cs.biu.ac.il

Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900, Israel

YANIV ALTSHULER

yanival@cs.technion.ac.il

Department of Computer Science, Technion, Haifa, 32000, Israel

ALFRED M. BRUCKSTEIN

freddy@cs.technion.ac.il

Department of Computer Science, Technion, Haifa, 32000, Israel

Published online: 26 September 2005

Abstract. Physical A* (PHA*) and its multi-agent version MAPHA* are algorithms that find the shortest path between two points in an unknown real physical environment with one or many mobile agents [A. Felner et al. *Journal of Artificial Intelligence Research*, 21:631–679, 2004; A. Felner et al. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 2002, pp. 240–247]. Previous work assumed a complete sharing of knowledge between agents. Here we apply this algorithm to a more restricted model of communication which we call *large pheromones*, where agents communicate by writing and reading data at nodes of the graph that constitutes their environment. Previous works on pheromones usually assumed that only a limited amount of data can be written at each node. The *large pheromones* model assumes no limitation on the size of the pheromones and thus each agent can write its entire knowledge at a node. We show that with this model of communication the behavior of a multi-agent system is almost as good as with complete knowledge sharing. Under this model we also introduce a new type of agent, a *communication agent*, that is responsible for spreading the knowledge among other agents by moving around the graph and copying pheromones. Experimental results show that the contribution of communication agents is rather limited as data is already spread to other agents very well with large pheromones.

Keywords: Mobile agents, A*, Shortest path, Pheromones, Multi-agent communication models.

1. Introduction

This paper introduces the notion of *large pheromones* as a model of communication and global knowledge sharing in multi-agent systems. With *pheromones*, agents communicate by writing and reading data at the nodes of the graph that constitutes their environment [1–3]. Unlike previous models of pheromones where only a limited amount of data can be written in each node, in the *large pheromones* model, there is no restriction on the amount of data that can be written in the nodes and thus each agent can write its entire knowledge in a node. We apply this model of communication to the multi-agent physical A* algorithm (MAPHA*) which is the multi agent version of Physical-A*

(PHA*). Both algorithms appeared in [4, 5]. These algorithms modify the A* algorithm [6] to find a shortest path in physical environments with one or many mobile agents that move around the environment and explore unknown territories. These algorithms are designed to minimize the travel effort of the agents.

Previous work on MAPHA* assumed a complete sharing of knowledge between agents [4]. In this paper, we modify the MAPHA* algorithm to use the pheromones communication model and explore the influence of the size of the pheromone on the performance of the algorithm. We show that increasing the amount of data that can be stored in each of the pheromones dramatically reduces the travel effort of the agents. The most important result of this paper is that with maximal usage of this model, i.e., with unlimited size of pheromones, the behavior of a multi-agent system is almost as good as with complete knowledge sharing between the agents. Another way of seeing this is that we compare between a system with a centralized shared memory to a system with distributed shared memory.

There are three main contributions in this paper:

- First, we introduce a new model of communication and we show that artificial pheromones can actually go beyond simple local information (e.g., classical routing table) as long as enough memory is available at the agent/node level.
- Second, we compare a multi-agent system centrally controlled and using global communication in a central shared memory with one in which decentralized agents decide individually about their action and can exclusively communicate with teammates through information stored and retrieved at the nodes and/or edges of the network. We show that the time performance of the later is almost as good the first even though it has no direct communication.
- Third, we provide a new version of the PHA* and MAPHA* family for a de-centralized multi agent system.

The paper is organized as follows. In the next section we first describe the PHA* and MAPHA* algorithms. Comprehensive discussions and all technical details on these algorithms are fully covered in the original papers [4, 5]. Section 3 discusses various communication models and Section 4 presents the paradigm of *large pheromones*. Section 5 provides comparison to related work. Section 6 describes the new versions of MAPHA* with large pheromones. Experiments are then provided in Section 7. Section 8 discusses the idea of communication agents. Conclusions and future work are presented in Section 9. More experimental results and different variations of the algorithms of this paper are presented in the Appendix. A preliminary version of this work appeared in [7].

2. Physical A*

The A* algorithm [6] is a common method for finding a shortest path in graphs that have exponential number of nodes (like combinatorial puzzles). A* keeps

an *open-list* of generated nodes and expands them in a best-first order according to a cost function of $f(n) = g(n) + h(n)$, where $g(n)$ is the distance traveled from the initial state to n , and $h(n)$ is a heuristic estimate of the cost from node n to the goal. $h(n)$ is *admissible* if it never overestimates the actual cost from node n to the goal. A* was proved to be admissible, complete, and optimally effective [8]. Therefore, any other algorithm claiming to return the optimal path must expand at least all of the nodes that are expanded by A* given the same heuristic h [8]. Usually, A* (or its variants) are applied to exponential combinatoric problems. In such domains, an A* expansion cycle is carried out in constant time as it takes a constant amount of time to retrieve a node from the open-list and to generate all its neighbors by applying domain-specific operators to the expanded node. Thus the time complexity of A* is usually measured in terms of the number of generated nodes [9, 10].

PHA* [4] modifies A* to find the shortest path in much smaller graphs which correspond to a real physical environment. Consider a mobile agent who needs to find a shortest path between two physical locations and assume that only a very small portion of the environment graph is known to the agent. Since A* is optimally effective, the mobile agent needs to activate the A* algorithm on this physical graph. For this type of graph, however, we cannot assume that expanding a node from the open list takes constant time. Many of the nodes and edges of this graph are not known in advance. Therefore, to expand a node that is not known in advance, a mobile agent must first travel to that node in order to explore it and learn about its neighbors. The cost of the search in this case is the cost of moving an agent in a physical environment, i.e., it is proportional to the distance traveled by the agent. PHA* expands all the mandatory nodes that A* would expand and returns the shortest path between the two points but is designed to minimize the traveling effort of the agent by intelligently choosing the next assignment of the traveling agent.

Note that since small graphs are considered here, we can omit the actual computation time and focus only on the travel time of the agent.

Unlike ordinary navigation tasks [11–14], the purpose of the agent in PHA* is not to reach the goal node as soon as possible, but rather to explore the graph in such a manner that the shortest path will be retrieved for future usage. In a navigation task, whenever the goal node is reached, the task ends. Most of the times the navigator did not travel via the shortest path and many times he does not know the shortest path. To identify a shortest path one must activate the A* algorithm and visit the set of mandatory nodes that A* would expand. Of course, our algorithm is designed to return the optimal (shortest) solution and thus we assume that the heuristics are admissible. For inadmissible heuristics we are not guaranteed to find the optimal solution so there is no need to activate a best-first search A*. In this case, any navigation algorithm will suffice to find a path that is not optimal. On the other hand, our problem is not an ordinary exploration problem [15], where the entire graph should be explored in order for it to be mapped out.

Another distinction should be made between the problem that we present here of finding a shortest path between two physical locations and the vast field of routing problems and algorithms [16–18]. We address the problem of finding

a shortest path between two physical points in a stable constant physical environment. Routing is usually the problem of finding paths in dynamic networks where the structure of the environment is dynamically changing due to load balancing and congestion. See [4] for more comparison to other algorithms and problems.

An example for a real application can be the following scenario. A division of troops is ordered to reach a specific location. The coordinates of the location are known. Navigating with the entire division through unknown hostile territory until reaching its destination is unreasonable and inefficient. Instead, one may have a team of scouts search for the shortest path for the division to pass through. The scouts explore the terrain and then report the shortest path for the division to move along in order to reach its destination. PHA* is an algorithm designed to help these scouts. Note that these scouts can be either human agents or artificial moving robots. In both cases they must explore the necessary portion of the physical environment in order to retrieve the shortest path between the two locations.

A* keeps global knowledge on the graph and only works properly (i.e., returns the shortest path) if the graph is stable throughout the search process. Thus, we assume that the graph is fixed and stable for a large portion of time but is not known in advance. Also, we assume that the graph has physical characteristics i.e., that traversing an edge is proportional to the cost of the edge. Thus, our algorithm and communication model are built to meet these assumptions. Telecom networks such as the Internet may not have these properties as they are usually not fixed since the load balancing of the graph is changing dynamically. Also, nodes of the graph might be added and deleted at any time. In that case, activating the A* algorithm is not relevant as global data in the open and closed lists are not accurate. Therefore, MAPHA* can not find shortest path in such environments and one of the routing techniques would be the algorithm of choice (see Section 5). However, if a network is fixed in its structure and load balancing then it may also serve as an application for our approach. See [4] for a deeper discussion.

2.1. Description of PHA*

We now turn to the description of the PHA* algorithm, focusing first on the case of a single mobile agent.

Nodes in the environment can be divided into *explored* and *unexplored* nodes. Exploring a node means physically visiting that node by the agent, and learning about it connecting edges and about the the exact identities and coordinates/locations of all its neighbors¹. PHA* activates essentially A* on the environment. However, in order to expand a node by A*, this node must first be explored by the agent so as to obtain the relevant data associated with it (i.e., incident edges, neighboring nodes and their exact locations and coordinates). PHA* works in two levels. The high level (which invokes the low level as a subroutine), acts like a regular A* search algorithm: at each cycle it chooses the best node from the open-list for expansion. New generated nodes

are evaluated and inserted to the open list of A* according to a cost function of $f(n) = g(n) + (h(n))$. $h(n)$ in our case, is the Euclidean distance between the new node n and the goal node. If the node chosen by the high level has not been explored by the agent, the low level, which is a navigation algorithm, is activated to bring the agent to that node and explore it. After a node has been explored by the low level it is expandable by the high level. If the chosen node has already been explored, or if its neighbors are already known, then it is readily expandable by the high level without the need to send the agent to visit that node. When a node is expandable, then the exact locations of its neighbors are known and thus we can calculate their Euclidean distance to the goal nodes and use it as their h value. The pseudo-code for the high level is given below.

```

high-level(open-list) {
.   while(open-list is not empty) {
.       target = best node from open-list;
.       if t=target is unexplored then {
.           explore(target) by the low level;
.       }
.       expand(target);
.   }
}

```

2.2. Low-level algorithms

The high-level algorithm, A*, chooses to expand the node with the smallest f -value in the open-list, regardless of whether or not the agent has already visited that node. If the chosen node has not been visited by the agent, the low level instructs the agent to visit that node. We call this node the *target* node for the low level. In order to reach the target node, we must use some navigation algorithm. In [4] a number of navigation algorithms are presented. The best ones are called A*DFS and I-A*DFS. Both these navigation algorithms attempt to navigate to the target via unexplored nodes. Thus, while navigating through unknown parts of the graph, the agent might visit new nodes that have not been explored yet and explore them on the fly. This may save the need to travel back to those nodes at a later time, should they be selected for expansion by the high-level algorithm.

At each step of the navigation to the target node, the agent should choose a node among its neighbors to travel to. Suppose the agent is at node v and is navigating to target. A*DFS chooses the neighbor n that minimizes

$$A^*DFS(n) = c(v, n) + d(n, \text{target})$$

where $c(v, n)$ is the cost of the edge (v, n) and $d(n, \text{target})$ is the straight line distance from n to target. We call it A*DFS since it uses a cost function which is similar to that of A*, i.e., $f(n) = g(n) + h(n)$.²

Improved A*DFS (I-A*DFS) enhances A*DFS as follows. Suppose that the agent is navigating to a target node. Along the way, it may pass near nodes that have a small f -value without visiting them, as they are not on the path

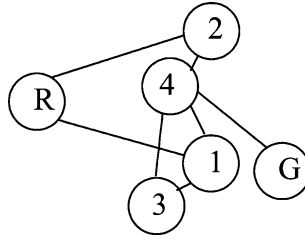


Figure 1. A*DFS vs. I-A*DFS.

to the target node according to the A*DFS navigation algorithm. Visiting such nodes when the agent is nearby, may save a lot of traveling effort in the future. I-A*DFS incorporates this notion. The basic concept of I-A*DFS is that while navigating to a target, the low level will select the next node to visit by considering not only its approximate distance from the target (as in A*DFS) but also the node's f -value. In order to achieve that, I-A*DFS chooses the neighbor n that minimize the following heuristic function:

$$c(n) = \begin{cases} A^*DFS(n) \cdot \left(1 - c_1 \left(\frac{f(T)}{f(n)}\right)^{c_2}\right) & \text{if } n \in \text{OPEN} \\ A^*DFS(n) & \text{otherwise.} \end{cases} \quad (1)$$

$A^*DFS(n)$ was described above. T and n denote, respectively, the target node and the neighboring node that is being currently evaluated. $f(\cdot)$ is the f -value of a node provided by the high-level A* and c_1, c_2 are constants, which, according to simulations, give the best results at $c_1 = 0.25, c_2 = 2$. For a better understanding of this formula the reader is referred to [4].

Consider the graph in Figure 1. Suppose that we want to find the path from node R to node G . Numbers inside the nodes correspond to the order in which standard A* expands these nodes. Assume that the agent activating PHA* traveled to node 1. The best node on the open-list is node 2 and it is set to be the target node. With A*DFS, the agent will navigate to 2 via node 4. With I-A*DFS, however, the agent will choose to first visit node 3 as it has a low f -value and it is very close to its current location. This will save the effort of navigating back to node 3 in the near future.

2.3. Improved high-level: WinA*

The basic idea behind I-A*DFS is that we give high priority to nearby nodes with low f -value even if they are off track in order to avoid future traveling to these nodes. The same concept can be used to improve the high level A*. The best-first order in which A* expands nodes from the open-list is optimal when the complexity of expanding a node is $O(1)$. However, in a real physical environment, where node expansion requires an agent to perform costly tasks, it is not always efficient to expand the current best node. Consider, for example, a nearby node that is not the best node in the open-list, but whose f -value is sufficiently small, such that with high probability it would be selected for expansion by A* in the next few iterations. An intelligent agent will choose to

explore such a node first, even though it is not currently the best node in the open-list.

In order to incorporate this capability into PHA*, *Window A** (WinA*) which is a generalized version of A*, was used. WinA* creates a set (window) of k nodes with the smallest f -values and then chooses one node from the set for expansion. The objective is to minimize the traveling effort of the agent, and not necessarily to reduce the number of expanded nodes. Thus rather than selecting only those nodes that have a small f -value, we choose also nodes that are sufficiently close to the location of the agent. In [4] it is shown that the best way of capturing these two aspects is by simply taking their product. Thus we order the nodes of the window by the cost function

$$c(n) = f(n) \cdot \text{dist}(\text{curr}, n),$$

where n is the node evaluated, $f(n)$ is its f -value, and $\text{dist}(\text{curr}, n)$ is the distance between n and the current location of the agent. We choose to expand the node with the smallest cost c^3 . In Figure 1 if the agent is located in node 1, it will choose node 3 as its new high level target since it has a small f -value and is close to its current location. The best performance was achieved in [4] by using WinA* for the high level and I-A*DFS for the low level.

2.4. MAPHA*: Multi-agent Physical A*

In [4], PHA* was generalized to the MAPHA* where a number of agents cooperate in order to find the shortest path. The task is that these agents should explore the necessary portion of the graph, i.e., the A* nodes as fast as possible. The assumption in [4] was that each agent can communicate freely with all the other agents and share data at any time. Thus any information gathered by one agent is available and known to all of the other agents. This framework can be obtained by using a model of a centralized supervisor that moves the agents according to the complete knowledge that was gathered by all of them. Another possible model for complete knowledge-sharing is that each agent broadcasts any new data about the graph to all the other agents.

MAPHA* also uses a two level framework. The high level chooses which nodes to expand, while the low level navigates the agents to these nodes. Since complete knowledge sharing is assumed there is one central high level which activates A*, (or actually WinA*). Suppose that we have p available agents and k nodes in the window of the front of the open list. We would like to distribute these p agents to the k nodes from the window as efficiently as possible. In [4] we described a distribution mechanism with the following attributes:

1. The distribution should favor assigning more agents to nodes in the front of the window, i.e. with a relatively small f -value.
2. The distribution should favor assigning an agent to a relatively close-by node.

For example, assume that two agents, A and B , are first located at node R of Figure 1. Nodes 1 and 2 are in the open list and the distribution mechanism

distributes the agents to nodes 1 and 2. Then, node 1 is expanded by the high level and nodes 3 and 4 are added to the open list. Now, nodes 3 and 4 are in the front of the window and the distribution mechanism now distributes agent A to target node 3 and agent B to node 4 so that these nodes can later be expanded by the A^* level.

Note that similar to the single agent PHA*, once an agent is assigned to a target node, it uses the I-A*DFS algorithm to navigate to that node. New nodes that are reached by the agent on the fly are immediately updated in the centralized databases obtained by high level. Here also, if such nodes are chosen for expansion, they do not have to be visited again.

In [4] it was shown that adding more agents is always efficient and reduces the time to solve the problem. As more agents were added, the search time converged asymptotically to the length of the shortest path. This means that asymptotically all paths from the initial state are traveled in a breadth-first search manner. This is to say that a sufficiently large team of agents is likely to produce a single agent that will travel along the actual shortest path with very little deviation from it.

3. Communication models and pheromones

There are many models for communication in multi agent systems. As described above, the most trivial model is complete knowledge sharing where any new discovery of an agent is immediately shared with all the other agents. Other models restrict the level of communication. Some models allow broadcasting or message exchanging between agents but restrict the amount of data that can be exchanged in each message or restrict the frequency or the number of messages that are allowed. Many times, a cost is associated with each message and the task is to solve the given problem while trying to minimize the cost of the messages involved.

Another distinction sometimes made (e.g. in the survey in [19]), is between implicit and explicit communication. Implicit communication occurs as a side effect of other actions, or “through the world” whereas explicit communication is a specific act designed solely to convey information to other agents on the team, e.g., broadcasting or message sending.

Additionally, it was discovered that in many cases, communication of even a small amount of information can lead to a great benefit [20]. It is this fact that stands at the basis of our proposed methods. More recent work in multi robot communication has focused on representations of languages and the grounding of these representations in the physical world [21, 22].

3.1. Communication by pheromones

In nature, ants and other insects communicate and coordinate by leaving trails of odor on the ground. The “data” placed on the ground is called *pheromones*. Studies on ants (e.g. [23–27]) show that the pheromone-based search strategies

used by ants in foraging for food in unknown terrains tend to be very efficient. It is believed that ants build a network of information with vertices represented by points of encounter between ants and the information is either passed between ants at a vertex with a physical encounter with other ants or via pheromone traces that are left on the ground. Inspired by nature, a famous and interesting model for communicating in multi-agent systems is that of ant-robotics (e.g. [1–3]). In this model, information is spread to other agents via *pheromones*, i.e., small amounts of data that are written by an agent at various places in the environment (e.g. vertices or edges of the graph) and can be later used or modified by other agents visiting that node.

In our ant-inspired model we assume that the information network is a graph, and the role of a pheromone is taken by a memory area on each node, that our search a(ge)nts can read and modify. This model is especially suitable for large networks, like the Internet, in which the nodes are powerful computers and the links have narrow bandwidth and are heavily loaded. This paradigm suggests a distributed group of one or more lightweight autonomous agents that traverses the network in a completely autonomous and parallelized way. Data is spread by the agents via these pheromones, which together serve as a distributed shared memory.

Note that considering the distinction raised above, a pheromone has both explicit and implicit attributes. Explicit in the fact that agents perform the action of writing in the pheromones. implicit in the fact that this is done through the environment.

Sometimes, the term pheromones is used to describe a distributed database that includes more than just spatially indexed information but also includes some sort of processing by the environment itself such as evaporation or propagation [17, 18]. Relevant information is reinforced while information that is becoming irrelevant evaporates or vanishes from the system.

In addition to pheromone based systems there are other behavior based systems which derives from the behavioristic regulations of insects from nature, e.g., systems that inspired from flocking and dispersing models [28–30] or predator-prey approach [31, 32]). More examples can be found in [33–35].

3.2. *The size of the pheromone*

What is the memory capacity of each pheromone? Biology scientists believe that pheromones in nature only include a very small amount of data due to the limited intelligence of insects. Nevertheless, for computerized multi-agent systems we can control the size and strength of artificial pheromones according to our needs. This is in fact one of the main purposes of our work as described below.

Although the exact size of the pheromone was not specifically quantified by the authors, many of previous works using pheromones assume that only a very small amount of data (no more than a few bytes) is written in each pheromone. The reason can be either because there exists a physical limitation on the amount of memory or because the intention of the authors was to keep the system as simple as possible and keep the agents and information passed as light as possible and yet to show the strength of these systems.

For example, in [3], when trying to explore a physical graph with non-communicating mobile agents, they assumed that the only fact that is written in a pheromone at a node of the graph is the outgoing edge that the last agent has chosen to leave that node. Thus, in that system when entering a node each agent resets the old pheromone and marks its own outgoing edge. Also, in [36] a team of cooperative cleaning robots are sent to clean a large “dirty” area in a grid which also contains obstacles. Each agent places a pheromone in each visited location. The pheromone only contains a stamp that it has visited that location. In the *ant colony optimization* system described below, when trying to solve the traveling salesman problem [37], also, the pheromone (placed on edges of the graph) only contained a number which is implicit measurement of the probability that the edge belongs to the optimal solution. In the AntNet system [16] described below, where ants randomly move around the network and modify routing tables, only data about the next hop is stored at each node.

It turns out, however, that despite these severe limitations on pheromone size, and assuming that only a very small amount of data is written in each pheromone, such agents are able to cooperate and achieve goals like covering a faulty graph [2], finding an Euler cycle in a graph [3] and solving various combinatorial optimization problems [38].

Pheromones usually serve as an indirect communication mechanism by using an evaporate/refresh process. This process typically does not require huge amounts of information. A small sized pheromone can only include local data and is not very suitable for problems such as our problem of finding a shortest path in a physical environment where a version of A^* should be activated and thus sharing of global data is needed. In the sequel we consider the effect of using larger pheromones, i.e., storing more data in the nodes, on the efficiency of multi-agent search.

4. Large pheromones

We suggest a new model of communication which we call *large pheromones*. Unlike conventional pheromones, we cancel the restriction on the amount of data that can be stored in each node, and consider the effect of this increased storage on the performance of the search algorithm. At the extreme, we assume that an agent can write its entire knowledge base (e.g. a complete list of nodes and edges known to that agent) at each visited node. With today’s hardware capabilities and computer architecture this is a reasonable assumption. With pheromones, we already assume that each mobile agent has the necessary hardware devices to allow reading and writing data in the physical environment. We also assume that there is a storage device at each node. Given that a storage device is installed at each node it is not reasonable to limit its size, since with current technology memory is very cheap in both cost and area. For example, it is not unrealistic to assume, say, one megabyte of memory at a node which can store a graph of tens of thousands of nodes. We can also assume that the time to read and write data from the large pheromones can be omitted when considering the traveling time of the agents. This additional data storage in the

large pheromones paradigm can help the agents in solving the problem faster and much more efficiently.

Large memory is not always available, e.g., in a system of nanorobots within a hostile environment, where only a very limited use of the environment is possible. Hence, we also consider a more modest memory capacity of the storage devices in the nodes. In that case, given the limited memory capacities and the entire knowledge base, we will address the question of selecting the most relevant portion of the knowledge for storing at the nodes. We will show below that when the size of the pheromone becomes larger and more data and knowledge can be written, the performance of the multi-agent system dramatically increases such that the most powerful large pheromone system is almost as good as a system with a complete knowledge sharing.

Note that a multi-agent system employing the large pheromone paradigm can be also called a *system with distributed shared memory* as opposed to a system with complete knowledge sharing which can be called a *system with central shared memory*. In both systems agents spread their knowledge by writing information in memory.⁴ The difference is whether there is one central location for that or whether this knowledge is distributed in the environment. In this paper we show that a distributed shared memory is almost as good as a single central shared memory.

4.1. Spreading data with large pheromones

With such large capacities of memory in each node, we present the following communication paradigm between the agents in general and in exploring unknown environments in particular. Each agent has a partial knowledge of the entire environment. It maintains a database with a partial graph that is known to it. Similarly, each node holds a database with a partial graph that is 'known' to it, i.e., knowledge that was written to it by the agents.

Data-merge Operation. Whenever an agent reaches a node it unifies (merges) the data known to it with the data that is written in that node. The agent then updates its own database as well as the database of that node to reflect the new unified data of view about the graph.⁵ We call this the *data-merge* operation. In this way data will be spread out very fast as long as agents visit many nodes in many areas of the graph and perform a *data-merge* operation at all the nodes that they visit. For example, assume that agent *A* visits node *n* and writes its knowledge in that node. After a while, agent *B* visits node *n*. Agent *B* will read the information in the node and will merge it with its own knowledge. After merging the knowledge, both agent *B* and node *n* hold the information gathered by both agent *A* and agent *B*.

If we assume a limited memory capacity of the nodes then simply unifying the data by data-merge operation described above is not always possible as the new (and thus larger) data might not fit in the limited amount of memory. If this is the case then the data merge operation is done in the following stages. First, the agent unifies its own data with the data that is written in the node and updates its own database to reflect the unified data. Now, the agent activates

a selection algorithm to determine the most informative portion of the data. Then, the agent replaces the previous pheromone with the new selected data. The selection algorithm is of course an independent part of the data merge operation and many variations are possible. As described below, we chose to select the nodes that are closest to the current node.

For the scouts example that we provided above, the large pheromones might be a graphical sketch of the known graph which with primitive resources can be drawn with the help of a piece of paper and a pencil. Of course, the scouts might choose to use sophisticated encryption techniques and more advanced memory storage devices, but the principle is the same.

5. Related work

The term *pheromone* is used by many authors in many algorithms and problem solving in multi-agent systems. Usually, it is used to represent a storage location which is dynamically modified during the activation of a multi-agent algorithm. The data in the pheromone is used for exchanging knowledge between the different agents. While this is similar for all systems, the term *pheromone* is used in many environments which can be fundamentally different. Pheromones might be used during the algorithm in many different ways and by different modifying protocols. In addition, the term *shortest path in graphs* is used by many authors in many different environments. In this section we try to clearly distinguish our work from others with regards to these aspects.

5.1. The AntNet system

A reminiscent approach to our idea of large pheromones and to the problem of finding a shortest path are many different solutions to routing in networks. For example, the heuristics in [17, 18] use ants to find shortest paths in (changing) telecom networks where the load balancing on the edges of the graph is not stable. Also, there are widely used Internet routing protocols such as BGP [39], RIP [40] etc.—all of which propagate shortest path information around the network and cache it at local nodes by storing routing tables with information about the next hop. Like in our model, in these settings a considerable amount of data is stored at nodes of the environment for the benefit of the agents. Another known routing system which uses “ants” and “pheromones” is the AntNet system [16]. In AntNet, ants randomly move around the network and modify the routing tables to be as accurate as possible.

While these approaches seem similar, there is a great difference between these works on routing and ours in the environments, the data that is stored at each node and the problem to be solved.

First, they assume that the environment is a telecom network that changes dynamically over time while we assume that the graph is fixed and corresponds to a real physical environment with Euclidean distances. Our problem is that this graph is not known in advance.

Second, these algorithms try to help an agent (which could be a phone conversation for example) to *navigate* to a goal location. In other words, given the destination node (which could be any node of the environment) and the knowledge written in the current node, these algorithms try to locally decide where should the agent go to next. Usually, the paths that they find are only sub-optimal. Our agents, on the other hand, are sent to explore the terrain in order to find the *shortest path* between given two locations for future usage. We only need to worry about these two locations and not about any other possible location of the graph. See Section 1 for a deeper discussion about the difference between our problem and a navigating problem.

Third, in these works, large routing tables with information about the next hop is being stored at each node while in our setting the large-pheromones store the entire knowledge about the structure of the complete graph and help the searching agents to make better choices of where to search next.

Fourth, since they assume that the graph is dynamically changing, the accuracy of the data stored by them might degrade over time. Thus, they put a lot of effort in keeping the data as accurate as possible at all times by having the ants consistently updating the routing tables. In such paradigm, the accuracy or time stamp of the pheromone needs to be analyzed by an agent that gets there in order to decide whether that data is still relevant. We, on the other hand, assume that the data placed at a node stays valid for a very large portion of time. This is reasonable for a stable unknown real physical environment.

Therefore, their approach solves a different problem of keeping routing tables as accurate as possible and will not be as effective as our algorithm for the problem that we discuss here of physically searching the graph for the *shortest path* between two given and fixed points.

In a sense, we can see our approach as generalization of next hop lookup tables by the fact that we provide partial graph information in each node, rather than just an instruction where to go next. In a sense, our large pheromones paradigm can be seen as a distributed blackboard system. In [41] they also use a distributed database which they also call pheromones.

The PHA* as well as MAPHA* with large pheromones that will be described in the next section, are deterministic algorithms that are designed to work only in a static environments where the structure of the graph is stable throughout the search. If the graph is dynamic and changes during the search then a probabilistic approach or one of the routing techniques would probably be a better choice.

5.2. *Ant colony optimization algorithms*

Another direction which uses the term *pheromone* is the *ant colony optimization* systems (ACO) which was first applied to the traveling salesman problem (TSP) [37, 42]. An ACO system include a set of cooperative agents cooperating to find good solutions to the TSP problem. The agents (also called “ants”) cooperate using an indirect form of communication mediated by pheromone they place on the edges of the TSP graph while building solutions. ACO systems were also used to solve other problems such as the sequential ordering problem [43],

quadratic assignment problem [44], vehicle routing problem [45] and other problems [45–50].

While ACO also use the terms *pheromones* and *ants*, again, there is a great difference between the ACO systems and our model of large pheromones in the environments and the data that is stored at each node.

First and foremost, the ACO systems are designed to work on environments with completely different fundamental assumptions. In our environment the ants are physical mobile agents that travel in a real physical environment which is a priori unknown. The ACO environment assumes that the complete graph is accepted as input and that a special attribute of the graph is needed, e.g., a TSP tour. They describe an algorithm for solving graph problems based on ants that are traveling along the graph. However, their ants are completely virtual and they are actually different processing units that can be implemented as parallel threads or tasks in the same CPU. Our environment and the platform where our algorithms take place is a real environment while their platform is a sophisticated data structure inside the computer’s memory representing a graph.

As described above, we assume that the *travel effort* across an edge is proportional to the length of the edge. The main target of our approach is to minimize the actual physical travel cost of the different agents while omitting CPU computational complexity. The *travel effort* concept has no meaning for the ACO systems as they are virtually traveling along a virtual graph inside the computer’s memory and they traverse an edge on a constant time. Therefore, from our point of view (of real physical environments and real physical mobile agents) the ACO systems are actually advanced distributed approaches that solve graph problems using sophisticated lookup tables (pheromones) which are updated by distributed threads or tasks (ants). In this sense the AntNet setting described above is closer to our settings since “real” agents traverse “real” environment only in AntNet the graph is a network of computers while our graph is a physical environment.

The ACO systems use an *evaporation effect* where the importance and accuracy of each pheromone degrades during time. This technically is more suited a for virtual environment inside the memory of the computer but is hard to be implemented in real physical environment. Here, we assume that pheromones are changed only by the activity of the agents.

Note again that the ACO systems assume that the complete graph is accepted as input and are designed for exponential time problems such as TSP etc. Therefore, the problem of finding a shortest path between two nodes is rather trivial in this setting as a simple algorithm such as Dijkstra’s algorithm can be activated (see [4] for a discussion about the difference between Dijkstra’s algorithm and A* and the different domains and assumptions which they are useful for).

6. MAPHA* with large pheromones

In this section we present our new version for MAPHA* where the large pheromones communication model is employed. In order to distinguish between

MAPHA* in the two communication models we will refer to the algorithm from [4] where *full communication* and complete knowledge sharing was allowed as *MAPHA*_{FC}* and to our new version where the *large pheromones* model is employed as *MAPHA*_{LP}*.

6.1. High level A* in MAPHA*

*MAPHA*_{FC}* was designed for the full knowledge sharing paradigm. Thus, the high level A* (or WinA*) was centralized and agents were sent to locally navigate to nodes by this centralized high level. When the large pheromones model is employed there is no such centralized entity and each agent runs the entire algorithm on its own. Thus, in *MAPHA*_{LP}* each agent activates the high-level A* (or WinA*) on its own, based on the partial knowledge of the graph that is known to it at any point of time. In a sense, each agent actually activates the single-agent version, PHA*. In other words, each agent keeps its own open-list of nodes and chooses to expand nodes from that open-list. As before, if the chosen target node has already been explored and thus all its neighbors (and their exact locations) are known to the agent, this node can be expanded immediately and its neighbors are added to the open-list. If, however, neighbors of the target node are not known to the agent, then, as with single agent PHA*, the agent will navigate to that target node with the help of a low-level navigation algorithm. In *MAPHA*_{LP}*, however, the large pheromones paradigm is employed. At each node that a navigating agent visits, it performs a *data-merge* operation. Its own knowledge is added to the node and knowledge about nodes that were not known to the agent is now learned by the agent. This might have a positive effect on the open-list and the high-level A* that is activated by this agent.

For example, suppose that agent *A* chose to expand node *t* from front of the open-list, and this node was not yet explored by agent *A*. Thus, agent *A* should now navigate to that node. On its way to *t*, agent *A* visits node *n*. Suppose that another agent, *B*, already explored node *t* and later wrote that data in node *n* while visiting it. When agent *A* reaches node *n* and performs a data-merge operation, it learns about node *t*. Therefore, agent *A* does not need to continue the navigation to node *t* and that node can be expanded by agent *A* immediately.

In order to derive *MAPHA*_{LP}*, the single agent version, PHA*, is therefore modified as follows. Whenever a data-merge operation is being performed, the graph that is known to the agent is modified. At that point, the high level A* is being invoked and the *f*-value of nodes is being updated. If the target node is now fully known to the agent, (i.e. the agent knows the neighbors of the target) then it is expanded immediately. In that case, a new expansion cycle begins and a new target node is being chosen.

In that sense, *MAPHA*_{LP}* and single agent PHA* are very much alike. They both activate A* (or WinA*) in the high level and send the agent to navigate and explore nodes in front of the open-list so that these nodes can be expanded. If these nodes are known then they can be expanded immediately and a new expansion cycle begins. The difference between them is the different circumstances in which nodes are known in advance and can be expanded even if the

agent is not physically located at these nodes. For PHA* this happens if the agent has already visited these nodes in the past. For $MAPHA*_{LP}$ this also happens if the agent learned about these nodes from data that was written in the pheromones by other agents.

If we only assume a limited memory capacity of the nodes then after the agent merges its data with data from the pheromone it will have to decide which are the most relevant nodes to write back to the pheromone and replace the old data of that pheromone. We have tried many variants and found out that the best performance was achieved by writing data about nodes that are closest to the current node. Thus, if for example the memory capacity allows 20 nodes to be written then the agent chooses the 20 nodes that are closest to the current node among all the nodes that are known to it (after unifying its own data with the previous data of the pheromone). The agent then replaces the previous pheromone with these 20 nodes.

6.2. *Initial distribution of the agents*

When activating more than one agent with this paradigm, the following problem arises. Since all agents activate the same deterministic algorithm and are initially located at the same node, they will necessarily make the same decisions and follow the same path. With $MAPHA*_{FC}$ [4] we solved this by simultaneously distributing the available agents to different nodes in the window which contains nodes that are in the front of the open-list. Therefore, they are located in different locations and later choose to navigate to different nodes. Note that an important attribute of PHA* in all its versions is that agents prefer to go to nodes that are physically close to them. If agents are located in different locations, they will, in general, make different decisions even if they have exactly the same knowledge about the graph.

In $MAPHA*_{LP}$, since every agent operates on its own, we have to make sure that at least at the beginning of the algorithm agents will go to different locations. We have considered a number of options that artificially distribute the agents at the beginning of the search. The results provided below use the following technique. All the agents start the search from the same node, but are dispatched at different times with a delay t between an agent and its successor. Thus, when an agent reaches a node, it may treat it differently from previous agents since it might know new data that was not known to them when they first arrived at that node, as the pheromone at the node has evolved. Thus, with this technique, agents start at the same node but continue differently due to the evolving of common knowledge stored in the pheromones.

6.3. *Static vs. dynamic data*

In the description above, pheromones within each node only included data about nodes and edges of the graph. This type of data can be called *static data* as it only includes topological data about the environment. This data will never change as we assume that the graph being explored is fixed throughout

the search process. With static data, if two agents arrive to a node (not necessarily at the same time) and they hold exactly the same information, they will choose to go next to the same node by activating the I-A*DFS low level algorithm described above.

To better treat this, we have enhanced the capability of the large pheromones to also include what we call *dynamic data*. The dynamic data part of the pheromone includes knowledge about the decisions and intentions of other agents. When an agent visits a node, then besides the data-merge operation of static data, it also writes facts about its current task. In particular, an agent writes the following facts in the dynamic data part:

- The neighboring node which it chose to visit next. This information corresponds to its current low-level navigation task.
- Which is the target node which it is going to. This information corresponds to its current high-level task.

When an agent visits a node then besides the data-merge operation which increases its knowledge it can also learn about the tasks of other agents and try to adjust its own tasks accordingly and try to behave differently. This can be done both in the high-level algorithm and in the low-level.

Handling dynamic data in the low level. We have tried many variations for handling dynamic data in the low level but only discuss the best variation here. Other variations are presented in Appendix A.1. In the best variation, the agents write their outgoing edge in the dynamic data part of the pheromone, similar to the EAW heuristic described in [3]. An agent will read the data about all the other agents and will give high priority to less traveled edges combined with other factors of I-A*DFS. Thus, an agent may choose an edge going to the opposite direction of the target node. Since this edge might have never been explored it may open some new opportunities and hopefully have a positive effect on the accumulation and spreading of information.

Handling dynamic data in the high level. Dynamic data can also improve the decision at the high level algorithm if each agent writes its current target node in each node that it visits. When choosing a target node we want the agent to consider knowledge about targets of other agents and give higher priority to other nodes. Also, until now, once a target node was chosen for an agent, it did not choose another target until the target node was expanded by its high-level (either by going there itself or by learning about that node on the fly). Here we want to add more points where the agent might consider again which target node it chooses to expand.

We have tried a number variations of using the dynamic data for deciding at different points which node to target. These variations tried different ways of giving low priorities to targets of other agents and combining this with the recommendation of the upper level A*. All these variations produced rather similar results. Thus, we omit the detailed description of these variations and focus on the best version. Other versions are covered in Appendix A.2. The most important

fact here is that when agent A realizes that some other agent B is already on its way to the same target, agent A reconsiders its plans according to the new information. It seems that any version that takes this dynamic data into consideration performs much better than only using static data as will be shown in the next section.

Our best version works as follows. Suppose that agent A is navigating to node t . On its way, agent A visits node n and finds out in the pheromone that another agent, B , is navigating to the same target node t (a special case of this is when an agent is first choosing a target node, see footnote below). In that case agent A stops executing the low level algorithm and invokes the high level algorithm. The motivation for that is that since agent B is already on its way to the same target then it is reasonable to believe that agent A will gain the information about this target node in the near future and thus agent A is better off choosing another target node. Thus, once this coincidence happens, agent A activates the high level window- A^* again and reconsiders which node from the window to choose as its new target. A while ago, node t was chosen to be the target of agent A . However, now, agent A is located in another location and might choose a different node as its new target given that someone else is navigating to node t . Note, that agent A might again choose node t as its new target even if other agents are already targeting it. In that case it may be assumed that this target is very important for the execution of the algorithm and is still the best target given the agent's new location and information.⁶

Note that changing the decision of which node to target has also some disadvantages. If an agent changes its target too often, it might enter a starvation situation in which it bounces back and forth between a number of targets without actually getting anywhere. Also, if agents change their minds too often then information from other agents regarding their targets might not be accurate as there is a chance that these agents changed their minds. Thus, this decision should be taken with care.

7. Experiments

We have implemented the variants described above and performed experiments on Delaunay graphs [51], which are derived from Delaunay triangulations. The latter are computed over a set of planar point patterns, randomly generated by a *Poisson point process* [51]. Points are distributed at random over a unit square, using a uniform probability density function. A Delaunay triangulation of a planar point pattern is constructed by creating a line segment between each pair of points (u, v) for which there exists a circle passing through u and v that encloses no other point. Such a triangulation can be characterized, in a sense, as one where each point is joined by a line segment to each of its nearest neighbors but not to other points. We have used the Qhull software package [52] to construct Delaunay triangulations (i.e., Delaunay graphs) over sets of points that were generated at random in a unit square. Figure 2 illustrates a 20-node Delaunay graph. The initial and goal nodes are picked at random after the graph is generated.

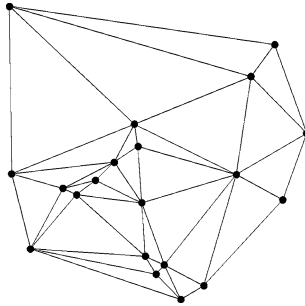


Figure 2. Delaunay graph of 20 nodes.

In principle, the characteristic whereby each node is connected to all its neighbors seems suitable for representing real road maps, which are the main object of our research. In practice, however, additional characteristics should be accommodated to more adequately capture a real road map. Thus we have also pursued *sparse* and *dense* Delaunay graphs that can be obtained from regular Delaunay graphs by random deletion and addition of edges, respectively.

Figure 3 illustrates the time elapsed as a function of the number of agents that were used to find the shortest path with different versions of MAPHA* on Delaunay graph with 500 nodes. Every data point (here and in all the other experiments) corresponds to an average of 250 different pairs of initial and goal nodes, that were picked at random. There are 6 curves in the figure. The bottom curve corresponds to the best version of MAPHA* with full communication, $MAPHA^*_{FC}$, from [4] and is used as a benchmark. Other curves show

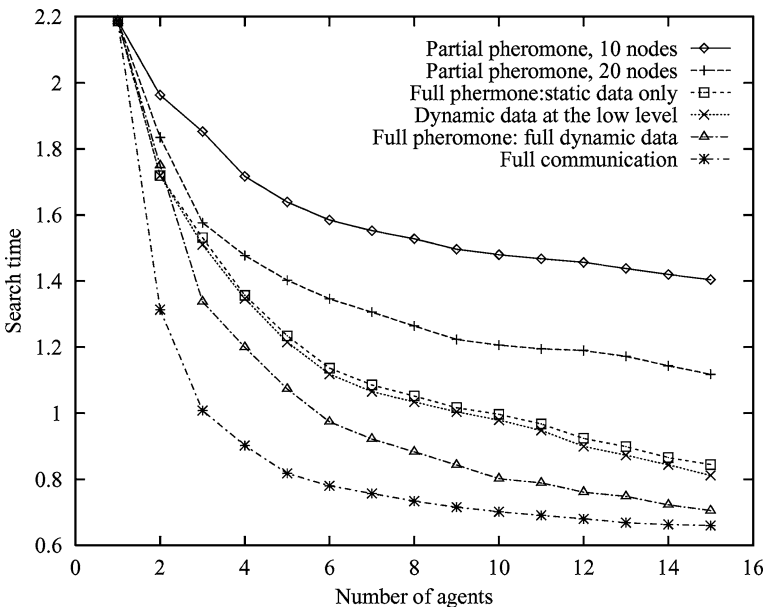


Figure 3. $MAPHA^*_{LP}$, 500 nodes.

the overall time cost of different versions of $MAPHA*_{LP}$. The top two curves show the behavior of $MAPHA*_{LP}$ where we only assumed a limited memory capacity at the nodes. In particular, in the top curve. Ten nodes were allowed to be written and the second curve allowed 20 nodes to be written at each pheromone. As explained above, the nodes selected to be memorized are those closest to the current node.

The rest of the curves assume unlimited data capacity and thus the entire graph can be written at each node. The third curve, “Full pheromones: static data only” shows the case where only static data was used. The next curve uses dynamic data for the low level only. Finally, the fifth curve, “Full pheromone: full dynamic data” uses the most powerful pheromone available, i.e., unlimited data capacity, static data and dynamic data at both the low level and the high level.

Note again that WinA* was used for the high level and I-A*DFS was used for the low level. Since we assume constant speed, the time is reported as the distance traveled by the agents until the solution was found.

The results show a clear phenomenon. As the pheromone becomes larger and includes more knowledge a significant improvement in the overall time is obtained. This parametric effect is achieved even though no explicit control is forced by a centralized supervisor. The figure clearly shows that all the versions that use the large pheromones paradigm with unlimited memory capacity keep most of the potential of the full knowledge sharing paradigm. Their performance is rather close to the performance of the full communication version. This means that with large pheromones, data is spread to other agents rather fast and it is almost as good as full communication and full knowledge sharing. This is true even for the simple version which includes static data only.

Dynamic data, with knowledge about routes, tasks and decisions of other agents further improves the performance. It seems that using it only in the low level did not significantly improve the case where only static data was used. However, adding dynamic data to the high level adds a lot of strength to this algorithm and this last version is almost as good as full communication model.

We have experimented with other sizes of graphs and on sparse and dense graphs and obtained similar results. These results are presented in Appendix B.

7.1. Experiments with no window

As described above, the best versions of the high level of both $PHA*$ and $MAPHA*_{fc}$ use what we called WinA* for maintaining the open-list. With WinA*, we define a *window* of the first K nodes of the open-list. Each agent chooses a node to expand from that window. This gives a lot of flexibility in the sense that a node can be chosen to be expanded even if it is not the best node on the open-list but it is rather close to the agent. This proved to be beneficial for both a single agent $PHA*$ and for $MAPHA*_{FC}$ [4]. We have also tried our new formulation of large pheromones without the window idea. This means that each agent must choose the best node from the whole open-list, as in pure A*. Figure 4 compares a version of $MAPHA*_{LP}$, when it used a window to a version when it did not use a window on a Delaunay graph with 1000 nodes.

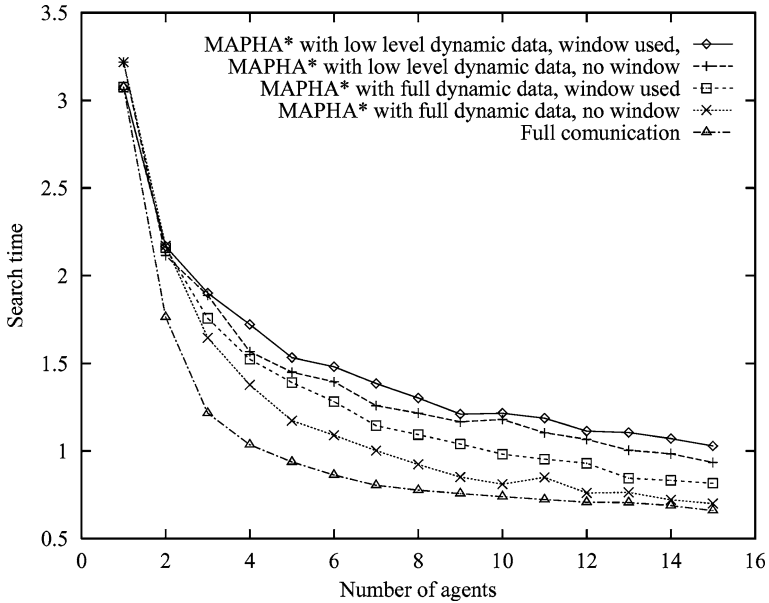


Figure 4. $MAPHA^*_{LP}$ with no window. 1000 nodes.

Surprisingly, $MAPHA^*_{LP}$ without the window enhancement (which is actually pure A^*) was better than using this enhancement. Note that this new version is almost as good as $MAPHA^*_{FC}$. The reason for this unexpected phenomenon is as follows. When the window size was large, agents were distributed by the high level to different target nodes in rather different areas of the environment. Thus, knowledge of other agents was possible only when an agent moved to other areas. With no window (or window of size 1) all agents were sent to the same target node probably via different navigating paths. Since all agents are navigating in the same area, knowledge of the current area was spread between the agents rather fast and then they all moved to another area. The scheme here is that all agents are sent to an area, explore it, share the knowledge among each other and then move to another area. This scheme of putting all the effort in one single area at a time seems to outperform a scheme of distributing the agent to different target nodes in totally different areas. This implies that with the large pheromones paradigm, data is spread more efficiently when the agents are rather close to each other.

In the full communication paradigm, a global knowledge-base was used and thus nodes in the window were identical to all the agents. Agents were spread to different nodes relying on both the available knowledge of all the known frontier areas of the graph and on the location of the different agents. Thus, the window approach proved beneficial as it helped to move agents to nearby nodes knowing that their new discoveries will be available to all other agents at once.

For the large pheromones paradigm, each agent holds its own open-list and consequently, its own window. Data is spread to other agents only when they visit the same nodes. Thus, applying the window approach was marginal. Note,

however, that even with the window approach, the overall performance of $MAPHA*LP$ is relatively efficient, as all the above figures show. Indeed, data was spread among the environment rather quickly and the speedup over a system with no communication at all (which is identical to a case where there is only one agent) was impressive for all variants that we used with this paradigm.

Experimental results for other sizes of graphs were similar and are presented in Appendix B.

8. Communication agents

In order to further improve the performance of our search algorithm, we also introduce another type of agents which we call *communication agents*. Similar to the *searching agents* described above, communication agents use the large pheromones communication paradigm. In other words, whenever they arrive at a node they perform the *data-merge* operation in that node. The difference between the two agent types is the algorithm that they activate in order to determine where to go next. As described above, the job of the searching agents was to gather new data from the search frontier. Thus, such an agent chooses to go next to some unexplored node. During its movement, the searching agent visits nodes and as a side effect performs data-merge operations and thus data is spread out to other agents. The primary task of the communication agent, is to spread data and thus perform as many data-merge operations as possible. Thus, a communication agent chooses to go next to a node where the contribution of the data-merge operation will be as significant as possible. As a side effect, if it arrives at a new unexplored node than this node is added to the general knowledge of the system. We have tried the following variants of decision algorithms which instruct the communication agent where to go next.

- *Random walk*: The simplest variant for the agent was perform a random walk (or patrol) in the entire environment.
- *Frontier walk*: In this variant we tried to guide the agent to move across the frontier border, thus meeting as many searching agents as possible in its travel.
- *Constant walk*: Here we forced the communication agent to move on a constant path from the frontier to the initial node knowing that if other communication agents do the same, data will spread out very efficiently.

We have experimented with many combinations of all variants of communication agents and different ratios of communication agents versus searching agents. Table 1 presents a sample of these experiments that were done with 15 agents. The first row, which serves as a benchmark is the case where all the agents were searching agents. The rest of the rows correspond to different types of communication agents where either 1, 2 or 5 searching agents were converted to be communication agents. The last column shows the relative time over the benchmark trial with 15 searching agents.

Table 1. Experiments with communication agents.

Algorithm	Searching agents	comm. agents	Relative time
–	15	0	1
Random	14	1	1.07
	13	2	1.15
	10	5	1.31
Frontier	14	1	1.06
	13	2	1.14
	10	5	1.28
Constant	14	1	1.08
	13	2	1.16
	10	5	1.45

Our experiments were rather conclusive and show that, surprisingly, given a number of agents, it is always beneficial to use them all as searching agents. Changing any number of agents to any variant of communication agents never reduces the overall cost. Converting more and more searching agents to be communication agents only increases the overall time.

At a first thought this is rather disappointing as we tend to think that communication agents can be very helpful. However, a deeper understanding of this phenomenon reveals the strength of the large pheromones paradigm and the power of this communication model. It seems that with this model of communication, searching agents are doing a very good job in spreading their data while searching and there is no need to invest any additional effort in data spreading by a special type of communication agent. Thus, any conversion of a searching agent to a communication agents only slows down the process as it would be better if this agent would explore new region of the environment and not concentrate on spreading data at the cost of no exploring.

8.1. Communication agents with breadth first search

In order to understand why communication agents do not contribute much, we made the following geometric observation. The search frontier of A* on a planar environment is usually formed as an ellipse. With MAPHA* we noticed that agents are exploring this ellipse in a rather fixed pattern of motion from the initial state to the goal state. Therefore, the usage of communication agents may not be beneficial here because agents are rather close to each other anyway. To check this hypothesis we tried to use communication agents when the high level A* was changed to simple breadth-first search. The search frontier in this case is a circle around the initial state and thus agents are much more distributed in the area. We found that even here, communication agents were not very beneficial. However, in this domain they were more productive than with the domain that used A* for the high level. While with A* they were never beneficial, with breadth-first search, there was one case where we got a solid improvement. This was when we converted a system with 15 searching agents to a system with 14 searching agents and a single communication agent. The

later case produced a reduction of around 5% in the overall time for all three types of communication agents.

Thus, we can conclude that the usage of communication agents is rather limited but it is also a domain dependent question of the geometrical structure of the environments and the paths and locations of the searching agents.

9. Conclusions and future work

We introduced the notion of *large pheromones* as a communication paradigm in multi-agent systems. We showed that in current technology this paradigm is reasonable and cheap to implement. We have used the PHA* algorithm as a test case for this paradigm. Results are encouraging as data is indeed spread out quite efficiently in all the different variations that we checked and the behavior of a multi-agent system is almost as good as with full-communication model. We have also showed that adding dynamic data about behavior of other agents greatly increases the efficiency of this paradigm, as agents make their decision based on a better knowledge of other agents. This fact is very important due to the modest memory requirements of writing this data.

The question is in what problem domains small pheromones are not sufficient and large pheromones are needed. We believe that large pheromones are needed in any domain where global data from different areas of the environment is critical to the decision making of all agents at all times. In that case smaller pheromones will not do the job. Our problem of activating A* in a physical environment is an example for this. Since A* expands nodes in a global best-first search order local data is not enough for this as shown in Figure 3.

For another example, consider a team of fireman agents that have to extinguish fire. The general geometrical structure of the fire is very important as it might cause agents to move to different locations. A counter-example might be a group of agents who are trying to explore and map unknown territories. Whenever an agent reaches a new node, it learns new valuable information. Thus, when locally realizing that there is a nearby unexplored area, moving to that area is always beneficial. Knowledge of other areas of the environments is not so crucial at every point of time. Similarly, the works in [2, 3, 38] need local data to improve their efficiency and thus very small pheromones were enough.

This paper is a first work on this new idea. Future work will proceed in the following directions.

- We have tried this concept on one particular problem. This paradigm should be applied to other problems. Indeed we are currently implementing these ideas to multi-agent fire detecting. Preliminary results look promising.
- In our problem, communication agents did not prove to be useful. Future work should determine whether there are other domains where communication agents are valuable.
- We provided a solid evidence of the benefits of large pheromones. A mathematical analysis of spreading data should be figured out, providing a better insights and bounds on achievable performance.

Acknowledgments

This research was partly supported by Israeli Ministry of Science Infrastructure grant No. 3-942. We would like to thank the anonymous referees for their enlightening comments and suggestions which greatly contributed to the quality of this paper.

Appendix A. Experimenting with different versions using dynamic data

In this section we describe different methods for handling the dynamic data in the low and high levels and provide experimental results.

A.1. Low level

As described above, in the low level an agent has to pick an outgoing edge to leave the current node to the target node. We would prefer to leave the node via an edge that has not been traveled before by other agents. While we have tried many combinations, here, we describe three versions for using dynamic data in the low level.

- *No dynamic data.* Here, no dynamic data is written in the pheromone. The agent exclusively chooses the outgoing edge according I-A*DFS.
- *Last agent.* In this version we keep a variable that stores the identity of one edge in the low-level dynamic data part of the pheromone. Each agent that leaves a node resets that variable and writes its own outgoing edge in that variable. When a new agent arrives at that node it will ignore the edge of the last agent and will choose the outgoing edge according to the recommendation of I-A*DFS among all the other edges. This will add diversity into the system as two consecutive agents will never leave via the same edge.
- *All agents.* Here we keep a counter for each edge counting the number of times an agent left this node via that edge. An agent will choose the less traveled edge, i.e., the edge with the smallest counter. In a case of a tie, I-A*DFS is consulted. This version has a great potential for saving future work as all edges will be traveled as time passes.

Figure A.1 presents the behavior of these different versions on a graph with 1000 nodes. The upper three curves correspond to these three versions when we only used the dynamic data in the low level. The lower three curves correspond to these three versions when we also used dynamic data in the high level. The figure show that all the three versions behave rather similar with a small advantage to the “less traveled edge” version which was therefore used in all our experiments presented in the main text. Note, that adding dynamic data in the high level gains a significant time reduction for all three versions.

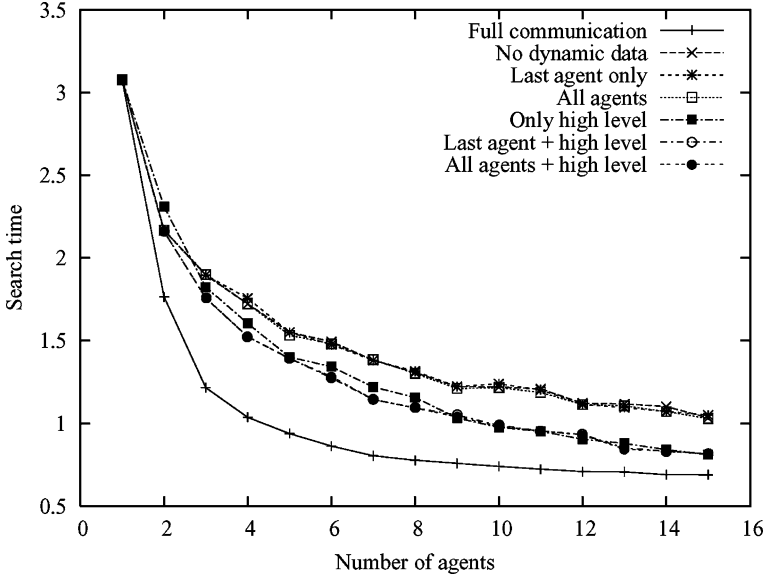


Figure A.1. The behavior of different versions of using dynamic data in the low level on a graph with 1000 nodes.

A.2. High level

As described above, in the high level an agent has to choose a target node from the open list. In *MAPHA*_{LP}* we might know the target nodes picked by other agents. We want to take that into consideration and give lower priority for such nodes when choosing target nodes. Also, if new knowledge about other agents is known we want the agent to reconsider its previous choice. We describe here three variations for doing this:

- *Old cost formula, at every node.* Here, we do not use the pheromone for dynamic data of the high level at all and do not consider information from other agents. At every node that the agent passes during the low level navigation it invokes the high level A* and chooses a new target. The new target node is the one with the best cost on the open-list according to the cost formula described in Section 2.3, i.e., $c(n) = f(n) \cdot \text{dist}(\text{curr}, n)$. The problem with this version is that it has a potential for starvation because an agent might bounce back and forth between a number of targets without actually getting anywhere. Nevertheless, we did not experience such episodes in our experiments.
- *Old cost formula, at key nodes.* Here, the high level is invoked only at *key nodes*. Key nodes are nodes where the agent realizes that another agent is also targeting the same target node.
- *New cost formula, at key nodes.* In this version, the high level is invoked only at key nodes. Here, however, we add another factor, $a(n)$, to the cost formula

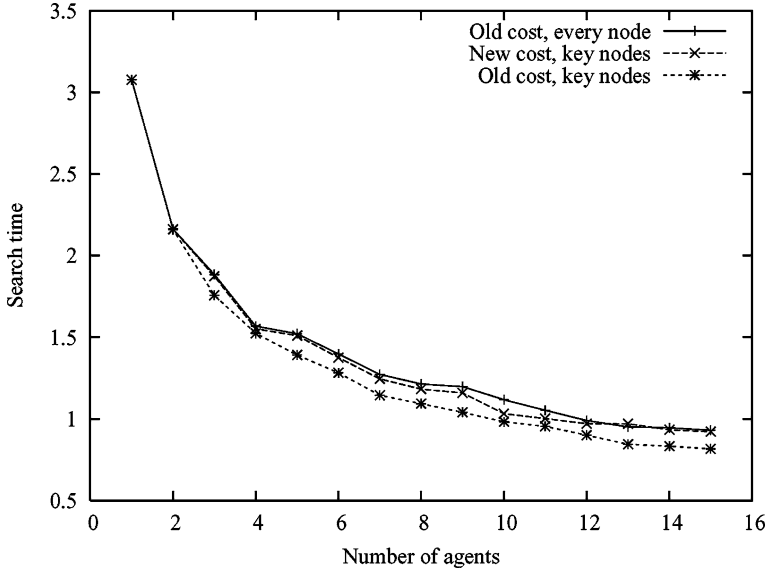


Figure A.2. The behavior of different versions of using dynamic data in the high level on a graph with 1000 nodes.

that chooses the target node and it is now -

$$c(n) = f(n) \cdot \text{dist}(\text{curr}, n) \cdot a(n)$$

The new factor, $a(n)$ counts the number of agents (including itself) known to it that are currently targeting the same node. Every time $a(n)$ increases (in key nodes), the high level is invoked, i.e., nodes in the window of the open list are order according to this cost formula and the node with the smallest cost is chosen.

Figure A.2 presents the behavior of these different versions on a graph with 1000 nodes. The best version is using the old cost formula at key nodes and it was used in all our experiments. This version probably has the best balance between keeping the same target and changing it. Unlike, the new cost formula, if other agents are targeting the same node, the agent has higher chance for keeping the same target. This means that in many cases it is not bad to target the same nodes as nearby agents can share information more easily. This same phenomenon was also valid when using a window of size 1 in Section 7.1.

Appendix B. Additional experimental results

Here, we provide additional experimental results for different variants of MAPHA* with large pheromones on Delaunay graphs of different sizes.

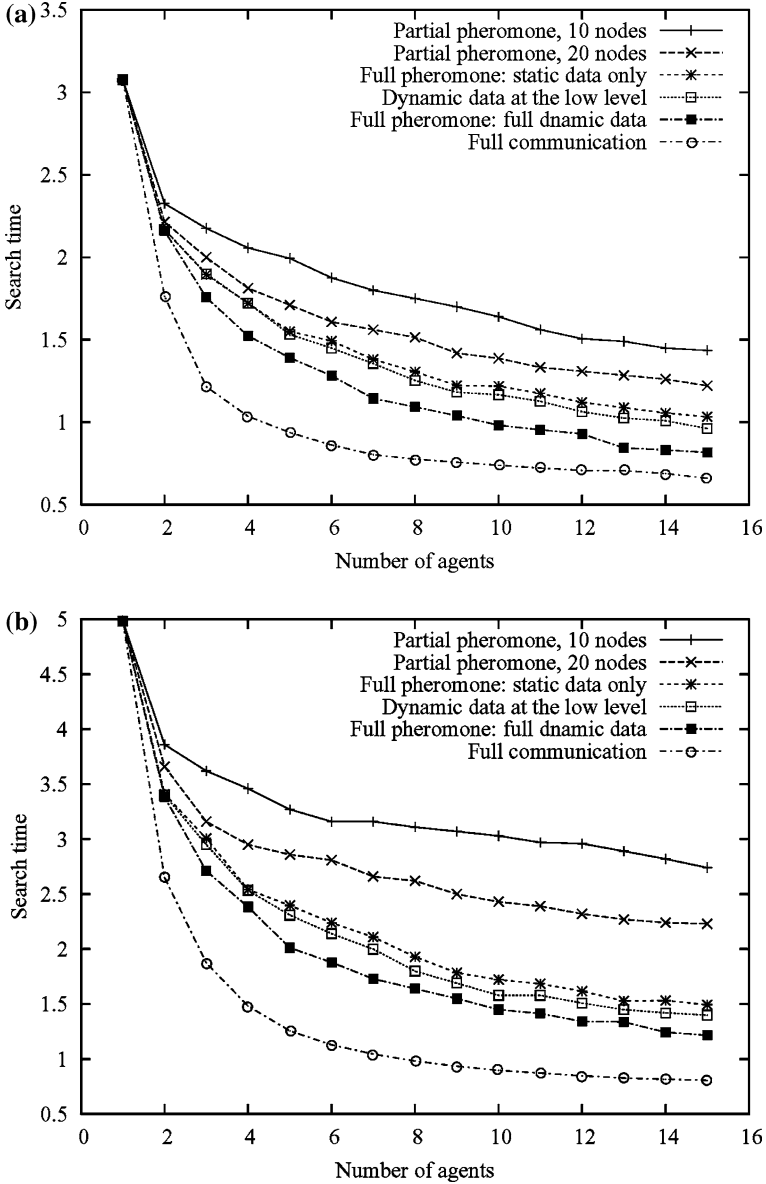


Figure B.1. Time consumption as a function of the number of agents for the various variants of MAPHA* on: (a) A Delaunay graph of 1000 nodes, and (b) a Delaunay graph of 2000 nodes.

Figures B.1a, b present the time costs of the various variants of the MAPHA* algorithm as a function of the number agents on Delaunay graph of size 1000 and 2000 respectively. The results confirm (similar to Figure 3) that as the size of the pheromone increases the overall travel time decreases. Again, the most powerful version is almost as good as the version with the full communication paradigm.

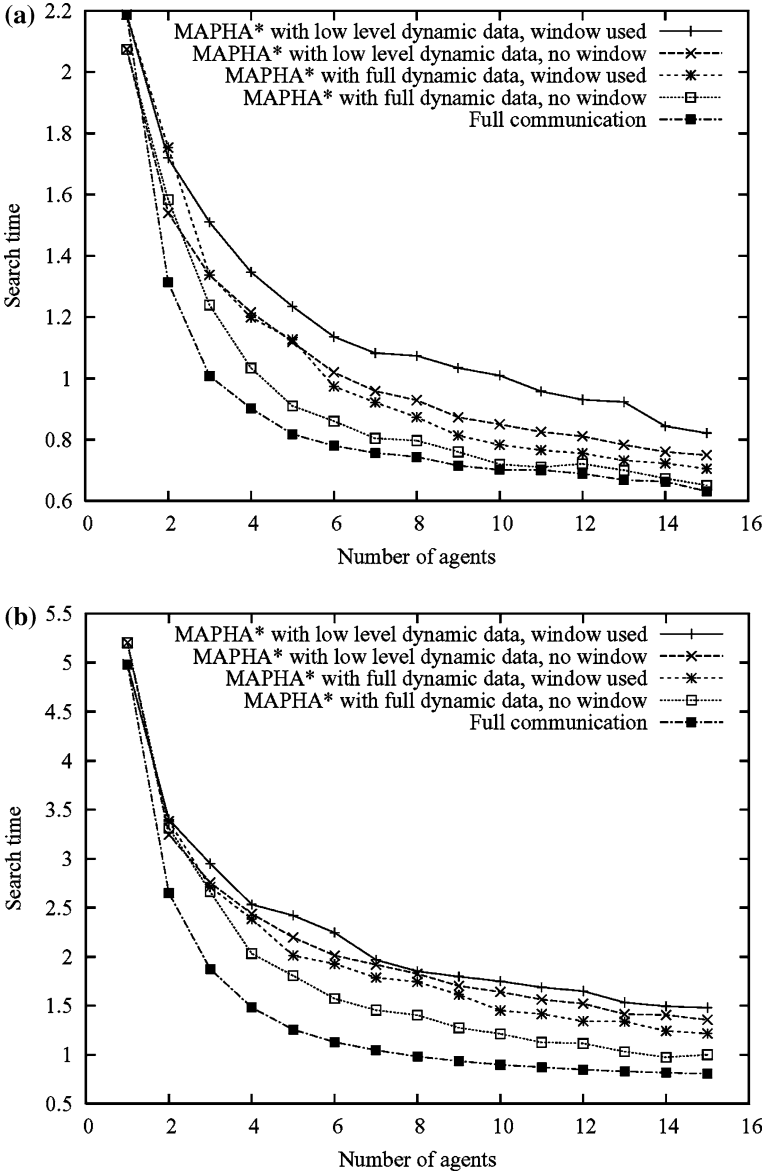


Figure B.2. Time consumption as a function of the number of agents for the various variants of MAPHA* with and without a window: (a) A Delaunay graph of 500 nodes, and (b) a Delaunay graph of 2000 nodes.

Figures B.2a, b present the phenomenon that $MAPHA^*_{LP}$ with a window of size 1 for the high level is better than the variation in which WinA* was activated for the high level of the algorithm. Here we present experimental evidence of that phenomenon on Delaunay graphs of sizes 500 and 2000. Here again, we can see that using pheromones greatly reduce the running time and the most

powerful version is almost as good as the version with the full communication paradigm.

Notes

1. As in [4] we assume a domain model that when a node v is visited by a search agent, the neighboring nodes and their exact locations (coordinates) are discovered, as well as the edges connecting them to v . See [4] for ways to deal with other models or assumptions.
2. Note, however, that this cost function is used here locally to find a path from the current node to the target node. This is different from the high-level A* which uses this cost function to find a path from the input initial state to the input goal state.
3. Combining this modified high-level variant with the low-level navigation creates some technical difficulties (like starvation), due to the fact that we no longer expand nodes from the open-list in a best-first order. See [4] for a comprehensive discussion and solution to this problem.
4. Of course, there are other methods that can achieve complete knowledge sharing but they are all logically identical.
5. Note that since both databases hold a valid list of nodes and edges then there are no conflicts between the two views of the agent and of the pheromone. Thus, the unification is done by simple adding of the two views and deleting duplicate copies of information.
6. A special case of this is when the agent is first choosing to target t . If the agent realizes that other agents are already targeting t , it should reconsider its decision and give t low priority. However, it turned out in our experiments that the best option is to stick with the recommendation of the upper level A* and not necessarily change the target at this point. See the Appendix A.2 for further details.

References

1. A. Wagner and A. M. Bruckstein, "ANTS: Agents, networks, trees, and subgraphs", *Future Gen. Comp. Sys. J.*, vol. 16, no. 8, pp. 915–926, 2000.
2. V. Yanovski, I. A. Wagner, and A. M. Bruckstein, "Vertex-ant-walk: A robust method for efficient exploration of faulty graphs", *Ann. Math. Art. Intel.*, vol. 31, no. 1–4, pp. 99–112, 2001.
3. V. M. Yanovski, I. A. Wagner, and A. M. Bruckstein, "A distributed ant algorithm for efficiently patrolling a network", *Algorithmica*, vol. 37, pp. 165–186, 2003.
4. A. Felner, R. Stern, A. Ben-Yair, S. Kraus, and N. Netanyahu, "PhA*: Finding the shortest path with A* in unknown physical environments", *J. Art. Intel. Res.*, vol. 21, pp. 631–679, 2004.
5. A. Felner, R. Stern, and S. Kraus, "PHA*: Performing A* in unknown physical environments," In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 2002, pp. 240–247.
6. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Trans. Sys. Sci. Cybernet.*, vol. SCC-4, no. 2, 100–107, 1968.
7. A. Felner, Y. Shoshani, I. A. Wagner, and A. M. Bruckstein, "Large pheromones: A case study with multi-agent physical A*," in *Forth International Workshop on Ant Colony Optimization and Swarm Intelligence. (ANTS 2004)*, 2004, pp. 366–373.
8. R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *J. Assoc. Comp. Machinery*, vol. 32, no. 3, 505–536, 1985.
9. R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search", *Art. Intel.*, vol. 27, no. 1, pp. 97–109, 1985.
10. R. E. Korf, "Linear-space best-first search", *Art. Intel.*, vol. 62, no. 1, pp. 41–78, 1993.
11. P. Cucka, N. S. Netanyahu, and A. Rosenfeld, "Learning in navigation: Goal finding in graphs", *Int. J. Pattern Recogn. Art. Intel.*, vol. 10, no. 5, pp. 429–446, 1996.
12. R. E. Korf, "Real-time heuristic search", *Art. Intel.*, vol. 42, no. 3, pp. 189–211, 1990.

13. L. Shmoulian and E. Rimon, "Roadmap-A*: An algorithm for minimizing travel effort in sensor based mobile robot navigation", In *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 1998, pp. 356–362.
14. A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994, pp. 3310–3317.
15. M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. P. Vadhan, "The power of a pebble: Exploring and mapping directed graphs," in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, Dallas, Texas, May 1998, pp. 269–278.
16. G. Di Caro, and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks", *J. Art. Intel. Res.*, vol. 9, pp. 317–365, 1998.
17. S. Appleby and S. Steward, "Mobile software agents for control in telecommunication networks", *Br. Telecom Technol. J.* vol. 12, pp. 104–113, 1994.
18. R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based load balancing in telecommunications networks", *Adapt. Behav.*, vol. 5, no. 2, pp. 214–237 1996.
19. E. Pagello, A. D. Angelo, F. Montesello, F. Garelli, and C. Ferrari, "Cooperative behaviors in multi-robot systems through implicit communication", *Robot. Auton. Syst.*, vol. 29, no. 1, pp. 65–77, 1999.
20. T. R. Balch and R. C. Arkin, "Communication in reactive multiagent robotic systems", *Auton. Robots*, vol. 1, no. 1, pp. 1–25, 1994.
21. L. Huges, "Collective grounded representations for robots," in *Proceedings of Fifth International Conference on Distributed Autonomous Robotic Systems (DARS 2000)*, 2000, pp. 79–88.
22. D. Jung and A. Zelinsky, "Grounded symbolic communication between heterogeneous cooperating robots", *Auton. Robots*, vol. 8, no. 3, pp. 269–292, July 2000.
23. F. R. Adler and D. M. Gordon, "Information collection and spread by networks of patrolling ants", *The Am. Nat.*, Vol. 140, no. 3, pp. 373–400, 1992.
24. D. M. Gordon, "The expandable network of ant exploration", *Anim. Behav.*, vol. 50, pp. 372–378, 1995.
25. B. Houmlldobler and E. O. Wilson, *The ants*, Springer, Berlin, 1990.
26. S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels, "Self-organized shortcuts in the argentine ant", *Naturwissenschaften*, vol. 76, pp. 579–581, 1989.
27. R. Beckers, J. L. Deneubourg, and S. Goss, "Trails and u-turns in the selection of the shortest path by the ant *Lasius niger*", *J. Theor. Biol.*, vol. 159, pp. 397–415, 1992.
28. M. J. Mataric, "Designing emergent behaviors: From local interactions to collective intelligence", in J. Meyer, H. Roitblat, and S. Wilson (eds.), *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Honolulu, Hawaii, MIT Press, 1992, 432–441.
29. J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario, "Self-organizing collection and transport of objects in unpredictable environments," in *Japan-U.S.A. Symposium on Flexible Automation*, Kyoto, Japan, 1990, pp. 1093–1098.
30. A. Drogoul and J. Ferber, From tom thumb to the dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Honolulu, Hawaii, 1992, pp. 451–459.
31. M. Benda, V. Jagannathan, and R. Doshiawalla, "On optimal cooperation of knowledge sources," Technical Report BCS-G 2010-28, Boeing AI Center, August 1985.
32. T. Haynes and S. Sen, *Adaptation and Learning in Multi-Agent Systems*, chapter Evolving Behavioral Strategies in Predators and Prey, Springer, 1986, pp. 113–126.
33. R. C. Arkin and T. Balch, "Aura: Principles and practice in review", *J. Exp. Theor. Art. Intel.*, vol. 9, no. 2/3, pp. 175–188, 1997.
34. R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE J. Robot. Automation*, vol. RA-2, no. 1, pp. 14–23, March 1986.
35. R. C. Arkin, "Integrating behavioral, perceptual, and world knowledge in reactive navigation", *Robot. Auton. Sys.*, vol. 6, pp. 105–122, 1990.
36. I. A. Wagner and A. M. Bruckstein, *Communications, Computation, Control, and Signal Processing: A Tribute to Thomas Kailath*, chapter Cooperative Cleaners: A Case of Distributed Ant-Robotics, Kluwer Academic Publishers, The Netherlands, 1997, pp. 289–308.

37. M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem", *IEEE Trans. Evol. Comp.*, vol. 1, no. 1, pp. 53–66, 1997.
38. V. Maniezzo, M. Dorigo and A. Colorni, "The ant system: Optimization by a colony of cooperating agents", *IEEE Trans. Sys. Man Cybernet.-Part B*, vol. 26, no. 1, pp. 29–41, 1994.
39. Y. Rekhter and T. Li, "A border gateway protocol," RFC 1771, T.J Watson Research Center IBM Corporation & Cisco Systems, March 1995.
40. G. Malkin, "RIPng protocol applicability statement," RFC 2081, IETF Network Working Group, January 1997.
41. P. Valckenaers, V. Marik, D. C. McFarlane, "Holonc and multi-agent systems for manufacturing," *First International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, 2003.
42. M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization", *Art. Life*, vol. 5, no. 2, pp. 137–172, 1999.
43. L. M. Gambardella and M. Dorigo, "Has-sop: An hybrid ant system for the sequential ordering problem", Technical Report IDSIA 97-11, IDSIA, Lugano, Switzerland, 1997.
44. L. M. Gambardella, E. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem", *J. Oper. Res. Soc.*, vol. 50, pp. 167–176, 1999.
45. B. Bullnheimer, R. F. Hartl, and C. Strauss, "An improved ant system algorithm for the vehicle routing problem," in *Sixth Viennese workshop on Optimal Control, Dynamic Games, Nonlinear Dynamics and Adaptive Systems*, Vienna, May 1997, pp. 21–23.
46. A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling", *JORBEL — Bel. J. Oper. Res., Stat. Comp. Sci.*, vol. 34, no. 1, pp. 39–53, 1994.
47. D. Costa and A. Hertz, "Ants can colour graphs", *J. Oper. Res. Soc.*, vol. 48, pp. 295–305, 1997.
48. R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based load balancing in telecommunications networks", *Adapt. Behav.*, vol. 5, no. 2, pp. 169–207, 1997.
49. G. Di Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks", *JAIR — J. Art. Intel. Res.*, vol. 9, pp. 317–365, 1998.
50. G. Navarro Varela and M. C. Sinclair, "Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation," in *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, Washington DC, USA, July 1999, pp. 1809–1816.
51. A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations, Concepts, and Applications of Voronoi Diagrams*. Wiley, Chichester, UK, 1992.
52. C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hull," Technical report, Geometry Center Technical Report GCG53, University of Minnesota, 1993.