Distributed Navigation in an Unknown Physical Environment

Arnon Gilboa, Amnon Meisels and Ariel Felner *
{gilboaar,am}@cs.bgu.ac.il, felner@bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel

Abstract. We address the problem of navigating from an initial node to a goal node by a group of agents in an unknown physical environment, where traveling agents must physically move around in the environment to discover the existence of nodes. As a result, agents need to travel in order to discover paths to the goal node. Agents communicate with a predefined set of neighbors by exchanging messages. A distributed algorithm, which is run independently by each agent, is presented. Given the current knowledge of the agent about the environment and the positions of other agents, the algorithm instructs the agent where to go next. The agent then updates its neighboring agents with its new position and its discovered nodes. An experimental evaluation of the algorithm is presented, with several different definitions of neighborhoods, on random physical graphs. Results show that the distributed intelligent behavior of agents generates spread of knowledge throughout the environment efficiently. Agents reach the goal node fast and the length of the path that they find is very close to that of the optimal path.

1 Introduction

We address the problem of navigation from an initial node to a goal node in an unknown physical environment, by a group of agents. The environment is represented by a partially-known weighted graph, where a weight of an edge is the Euclidean distance between the two nodes it connects. Only when an agent visits a node, the weights of outgoing edges and the neighboring nodes are revealed. In other words, agents must move around the graph in order to find information about other nodes and about weights of edges.

Let us start by making a distinction between navigating to a node and finding the optimal path from an initial node to a goal node. When navigating, the task is to reach the goal node as soon as possible. At every step the task of the navigator is to find a path from the current node to the goal node. Once the navigator reaches the goal, the task is accomplished. Many times, the navigator does not travel via the shortest path and does not even know the shortest path between the two nodes upon arriving to the target node.

Much of the literature on navigation deals with physical mobile robots that move in a real environment. The published research focuses usually on the issue of assisting the

* Department of Information Systems Engineering

robot to recognize physical objects in its environment. We refer the reader to [Bender *et al.*, 1998; Burgard *et al.*, 2005] which contain extensive surveys of various related approaches and state of the art techniques. The present work is different, as it is represents the physical world by a discreet abstract graph with clear definitions of nodes and the graph is initially unknown.

Finding an optimal (shortest) path is different than navigation. Here, even if we have some path from the initial node to the goal node, we must perform a systematic search on the graph in order to guarantee its optimality. A common method for finding the shortest path in graphs is to use the A* algorithm [Hart, Nilsson, & Raphael, 1968]. A* keeps an *Open-list* of generated nodes and expands them in a best-first order according to a cost function defined by $f(n) = g(n) + h(n)$. In standard A* $g(n)$ is the distance traveled from the initial node to node n, and $h(n)$ is a heuristic estimate of the cost of traveling from node n to the goal node. $h(n)$ is *admissible* if it never overestimates the actual cost from node n to the goal (e.g. Euclidean distance between the two nodes in a physical environment).

Physical A* (PHA*) and its multi-agent version MAPHA* [Felner *et al.*, 2004] are algorithms that find the shortest path between two points in a partially known real physical environment with one or many mobile agents. These algorithms modify the A* algorithm to graphs with physical characteristics where agents need to physically travel to a node in order to learn about its neighbors. In such graphs, in order to expand a node, a mobile agent needs to physically travel to that node. PHA* and MAPHA* tried to minimize the total travel cost of the agent(s) in the environment. In MAPHA* they assumed a complete sharing of knowledge between agents - any new discovery of an agent is immediately shared with all the other agents. This could be achieved by either a supervisor agent who coordinates the agents or by global broadcasting the knowledge to a shared blackboard.

The present paper addresses the navigation problem in partially known physical graphs (described above) by a distributed group of agents. The task is to reach the goal node as soon as possible. When the first agent reaches the goal the task is achieved and the agents halt. A distributed navigation algorithm, DisNAV, for a partially known physical environment is presented. Each agent runs the same algorithm that uses its own partial knowledge of the environment. A distributed algorithm is in general more fault tolerant and is expected to reduce communication, compared to a centralized algorithm [Lynch, 1997].

There are many models for communication in multi agent systems so that agents can cooperate and coordinate their decision making. 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. The present study focuses on a distributed navigation model.

Agents exchange relevant information, such as their current position and distances to neighboring nodes. In each step, agents receive messages from other agents, decide whether and where to move, and send updates to their neighbors. Each agent can only communicate with a predefined set of agents - its *neighbors*. The *neighbor* relation is symmetric. To be realistic, agents are constrained to communicate with their neighbors only if the distance between them is below a given *radius*. This *distance constraint* enforces neighboring agents to stay close to one another at all times.

The selection of which node to go next is done independently by every agent, based on information gathered by itself and by its neighbors. Note, that in this paradigm, each agent that arrives at the goal node has its own view about a path from the initial node to the goal node.

Two measures of performance are used. One measure is the path traveled by the first agent arriving at the goal node. The other measure is the quality of the best path found, compared to the optimal path. Our agents try to navigate and find a path to the goal as fast as possible in order to minimize their travel effort. They also attempt to explore neighboring nodes, in order to discover a path that is closer to the optimal.

These two measures might be in conflict with each other. This is known as the exploration versus exploitation conflict. The present study combines these two aspects and tries to find the best balance between them. The influence of the number of neighbors that each agent has on the overall performance is studied experimentally. We implemented the DisNAV algorithm and experimented on different graphs and different number of agents and neighbors. Results show that the proposed distributed navigation algorithm generates an efficient spread of knowledge throughout the environment. Agents reach the goal node very fast and the length of the path that they find is close to that of the optimal path.

2 Distributed Navigation Algorithm

The main motivation of the present study is to look at multi-agent navigation as a distributed problem, where agents communicate only with their neighbors. In order to simplify the definition of the distributed navigation problem, the present study uses a static neighborhood. Static neighborhoods enables an easy definition of a static connected communication network of agents. Each agent knows all its neighbors in advance, so it does not need to know the distance from agents which are not its neighbors. Static neighborhoods can cause some restriction on agent moves. The distance constraint may prevent agents from making some of their possible moves, if they move the agent too far from its neighbors. Of course, an agent cannot communicate with a nearby agent which is not its neighbor.

The distributed algorithm presented here, DisNAV, is run by each agent independently. Agents communicate with their neighbors, exchanging information such as their current position, node discoveries and move proposals. Agent navigation is based on its local information which is updated according to the received messages. Agents keep local information in three main data structures: *KnownGraph*, *Neighbors* and *Open-list*, all of which are described below.

– **KnownGraph** - The *KnownGraph* is the part of the world currently known to the agent. *KnownGraph* is a sub-graph of the entire environment modeled as a weighted graph. We assume that all agents start from the same initial node and agents send each node discovered to their neighbors. Therefore, *KnownGraph* will always be a connected graph. During the execution of the navigation algorithm, *KnownGraph* is used by each agent for distance estimation to optional nodes exploration. In addition, when an agent reaches the goal node, *KnownGraph* is used for finding the best candidate path between the initial node and the goal node.

– **AgentView** - The *AgentView* data structure, contains the *state* of the neighbors of the agent. *AgentView* is updated by the latest message received from each of them. The *state* of a neighboring agent includes its current position and its current target or move proposal.
– **Open-list** - Each agent maintains a local *Open-list* that includes nodes discovered during navigation, by itself or by neighboring agents. Every agent expands the nodes in its *Open-list* in a best-first order according to a cost function. The cost function includes an intelligent heuristic function that will be described below.

We now present the details of the DisNAV algorithm. The pseudo-code for the algorithm is given in Figure 1. The navigation is done by *moves*. First an agents picks a *target* node from its Open=list which it believes that will get him toward the goal. A *move* is a sequence of *steps* from the current node of the agent to the target node, where each step includes a trip of the agent between two adjacent nodes. Each proposed move should be approved by all neighbors before its actual performance. The main procedure ran by each agent is ***DisNAV***. DisNAV, is a message driven algorithm. In each *cycle* of DisNAV an agent reads all the received messages (lines 5-6) and decides about a proper resulting operation. Resulting operation can be a message sending (e.g. NODE, APPROVAL etc.) and can also include a performance of an approved move. An agent running the algorithm can be in one of several possible logical states. Different events can change the state of an agent. The initial state of an agent is READY. Each agent runs a loop, which ends when the agent enters the TERMINATED state.

The basic state is the READY state. In that state the agent selects a target node from the Open-list to navigate and send a *move proposal* message to its neighbors. If the move is approved by all the neighbors then the agent performs the move by navigating to the target node. An agent considering a move proposal (***proposeMove***, lines 51-54), selects the best node from its *Open-list*. The selection is performed according to a cost function which is described below. After the best node b is selected, the agent sends a move proposal to navigate to b. The cost function ensures that moves that break the distance constraint between the agent and its neighbors will not be selected by giving them a cost of infinity[1].

After a move is chosen (lines 52-54), the agent sends an OK? message proposing the move to its neighbors. The state of the agent is changed to WAIT_FOR_APPROVAL. Receiving an OK? message (lines 19-23), an agent has two choices. It can either approve the proposal, replying with an APPROVAL message or reject the move using a NOGOOD message. We say that move proposals of two neighboring agents are in *conflict* if performing them both at the same time will break the distance constraint between them. Such a conflict is resolved in favor of the move with the lower cost. In such case, a NOGOOD message is sent by the agent with the better move.

Whenever an agent receives APPROVALs from all its neighbors (lines 25-28), its state changes to IN_TRANSIT. In this case the target node is removed from the *Open-list*, and the agent starts performing the move to the target. Otherwise, if an agent receives a NOGOOD (lines 29-31), the state returns to READY and the move proposal

[1] To optimize the overall performance, sometimes agents decide to remain idle, if the contribution to the group effort is low. For example, sometimes it is better to wait until a neighboring agent performs a low cost move which adds important knowledge to the agent's *KnownGraph*

is canceled. A move includes a sequence of steps. At each step the agent chooses the neighbor w that minimizes the sum of the distances from the current node v to w and from w to the target node t. This cost function is called A*DFS in [Felner *et al.*, 2004] since it uses a cost function which is similar to that of A*, i.e., $f(n) = g(n) + h(n)$. Note, that this is a cost function is used here locally to find a path from the current node to the target node.

Upon agent arrival to a node (line 37), the procedure ***uponArrivalToNode*** (lines 40-50) is called. If it is the goal node, the agent changes its state to TERMINATED and sends a TERMINATE message. Each agent receiving such a message (lines 32-34), sends it to its neighbors and changes the state to TERMINATED. Otherwise, if the node is not the goal, the agent learns about this node's neighbors and sends a NODE message, to inform the neighbors about its current position and about the node's neighbors. If the current node is in the *Open-list* this node is expanded, i.e., it is removed from the *Open-list*, and the node's neighbors which where not visited yet, are inserted to the *Open-list*. If the current node is the target node (lines 48-50) the state returns to READY and the agent considers another move proposal.

Agent receiving a NODE message (lines 8-18), an agent updates its *AgentView* with the sender location and updates its *KnownGraph* with the new nodes and edges discovered. In case the node is in the *Open-list*, it is immediately expanded by the agent without the need for the agent to physically visit that node. This is the main contribution of data sharing among neighboring agents.

3 Navigation and Exploration Heuristics

Each agent running the DisNAV algorithm, owns a local *Open-list*. The *Open-list* includes nodes discovered during navigation, as well as nodes explored by neighboring agents. The nodes are chosen to be targets by the agent in a best-first order according to a cost function[2].

The ordinary cost function used by A* and other search algorithms for finding optimal paths is $f(n) = g(n) + h(n)$, where $g(n)$ is the distance traveled from the initial node to the node n, and $h(n)$ is a heuristic estimate of the cost of traveling from node n to the goal node. Real-Time-A* (RTA*) [Korf, 1990] selects the node with the best cost in the search frontier. The problem solver then moves one step along the path to that node, and the search continues from the new state of the problem solver. The cost function for an arbitrary node n in RTA* is also $f(n) = g(n) + h(n)$ but $g(n)$ is an estimation of the distance of node n from the current node and not from the initial node.

DisNAV deals with physical graphs where nodes can be expanded only after an agent physically visits them. This is completely different from RTA* which assumes that a node can be expanded in the computer's memory without an agent having to physically visit that node. However, the cost function used in DisNAV treats $g(n)$ similarly to RTA* and is measured according to the current location of the agent. Thus, at every step the cost values of the entire *Open-list* are changed so that g will reflect the current location. The best node on the open list according to the cost function is chosen and becomes the target node for that agent.

[2] When an agent knows that one of its neighbors is navigating to a target node n then n can be removed from the *Open-list*, so the agent will not choose the same target

DisNAV:

```
1.  state ← READY; node ← initial; proposal ← null
2.  expand node
3.  update KnownGraph and Open-list
4.  while(state ≠ TERMINATED)
5.    while(not msgQueue.isEmpty)
6.      msg ← msgQueue.getFirst
7.      switch msg.type
8.        NODE:
9.          update KnownGraph, AgentView and Open-list
10.         if(state=READY)
11.           proposeMove
12.         else if(state=WAIT_FOR_APPROVAL)
13.           if(move proposal is redundant)
14.             state ← READY
15.             ignore proposal
16.           else if(move proposal should be updated)
17.             update proposal
18.             send(OK?,neighbors)
19.       OK?:
20.         if(self already sent a better conflicting proposal)
21.           send(NOGOOD,sender)
22.         else
23.           send(APPROVAL,sender)
24.       APPROVAL:
25.         if(state=WAIT_FOR_APPROVAL and all approved)
26.           state ← IN_TRANSIT
27.           target ← proposal
28.           remove target from Open-list
29.       NOGOOD:
30.         if(state=WAIT_FOR_APPROVAL)
31.           state ← READY
32.       TERMINATE:
33.         state ← TERMINATED
34.         send(TERMINATE,neighbors)
35.   if(state=IN_TRANSIT)
36.     move to a neighboring node, navigating to target
37.     uponArrivalToNode
38.   else if(state=READY and proposal=null)
39.     proposeMove
```

uponArrivalToNode:

```
40. if(node=goal)
41.   state ← TERMINATED
42.   send(TERMINATE,neighbors)
43. else
44.   discover node neighbors
45.   send(NODE,neighbors)
46.   if(node in Open-list or node=target)
47.     update KnownGraph and Open-list
48.     if(node=target)
49.       state ← READY
50.       proposeMove
```

proposeMove:

```
51. proposal ← get best node from Open-list
52. if(proposal is worthwhile)
53.   send(OK?,neighbors)
54.   state ← WAIT_FOR_APPROVAL
```

Fig. 1. The DisNAV Algorithm

A good cost function for our purpose needs to combine both navigation and exploration. In other words, we want to get to the node as fast as possible but also to learn about new regions of the graph in order to provide useful information to other agents. For this one can use combinations of the following two functions:

- $f(n) = g(n) + h(n)$ where g is measured from the current node and h is the Euclidean distance from n to the goal. This is the navigation part which leads the agent to the goal as fast as possible.
- $d(n)$ which measures the distribution of the neighboring agents in the environment. It returns the average distance of a node n from neighboring agents. Larger $d(n)$ means that the neighboring agents are relatively not gathered around node n.

Nodes with small f-value are preferred, but on the other hand, so are nodes with large d-value. Such nodes are on a promising path to the goal and are distanced away from other agents. A number of combinations between f and d were tried. The best results were obtained by using:

$$c(n) = f(n)/maxF + p \cdot (1 - d(n)/maxD)$$

where $maxF$ is the largest f-value on the *Open-list* and $maxD$ is similarly the largest d-value over all nodes of the *Open-list*. The constant p, defines the weight of the exploration in the cost function. Our extensive empirical studies have shown that $p = 0.3$ produces the best performance.

4 Experimental Evaluation

DisNAV was evaluated experimentally on random Delaunay graphs [Okabe, Boots, & Sugihara, 1992] which are common testbeds (e.g., [Felner *et al.*, 2004]) for simulating physical graphs. They are derived from Delaunay triangulations which are computed over a set of planar point patterns, generated by a *Poisson point process* [Okabe, Boots, & Sugihara, 1992]. Points are distributed at random over a square, using a uniform probability density function. In a regular Delaunay graph, each node is connected to all its neighbors.

This property may not always apply to a real road map. For example, nearby geographic locations may not always be connected by a road segment, due to the the existence of obstacles like a mountain or a river. To capture this additional characteristic, sparse Delaunay graphs were considered. Instances of these variants can be easily obtained from regular Delaunay graphs by random deletion of edges. We define the *density* as the fraction of Delaunay edges left in the graph. The graphs were generated to be connected, by first adding a spanning tree edges and then adding random Delaunay edges up to the desired *density*.

The first set of experiments compares different densities on sparse Delaunay graphs with 1000 nodes, on a square with 100x100 units. We have generated 100 different graphs and selected a pair of nodes with maximal distance as the initial and goal nodes. Graph densities of 0.3, 0.35 and 0.4 were used. The neighbors of the agents were defined as follows. The agents were ordered in a chain and each agent was defined to be a neighbor of its predecessor and successor in the chain. Thus, each agent (except for the

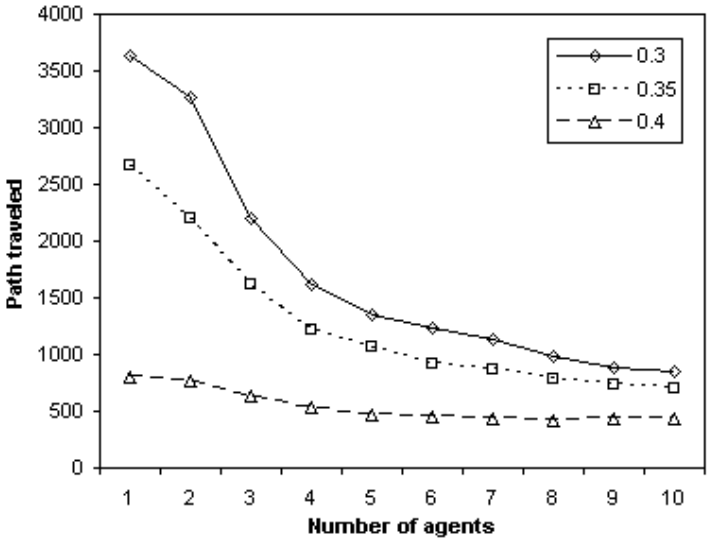

Fig. 2. DisNAV best agent. 1000 nodes

agents at the two ends of the chain) had two neighbors but no 3-agent cliques existed. The maximal distance between neighbors was set to 75 units. The number of agents in the experiments was varied from one to ten. In the case of a single agent, the d function is not defined and only the f function is used.

Figure 2 presents the length of the path traveled by the first agent reaching the goal node, as a function of the number of participating agents. As the number of agents grows, the first agent reaches the goal node after traveling a shorter path. This is because data is shared among the agents efficiently. It is also clear, that as the density of the graph grows, the path traveled by the first agent is shorter. In graphs with high densities, agents have more alternative paths to the goal node and there are only a few deadends. Such graphs are easy for navigating and agents can find their way by simple directional navigation. They do not need a lot of information from their neighbors. Thus, the improvement for adding more agents is small and a single agent is almost as good as many agents. When the graph is sparse and there are many deadends, the distributed algorithm outperforms a single agent. In other words, the first agent from the group reaches the goal much earlier than a single agent.

Figure 3 presents the ratio of the length of the best path found by the first agent reaching the goal node to the optimal path. It is clear from figure 3 that as the number of participating agents grows, the first agent to reach the goal node finds a better path, compared to the optimal. In graphs with low densities (0.3 and below) the best path found is actually the optimal path. This is an interesting result. In such graphs agents cover a much larger portion of the graph due to deadends, so on arriving to the goal node, their *KnownGraph* covers a larger portion of the complete graph. In other words, the higher cost needed to reach the goal node (figure 2) generates a path that is closer

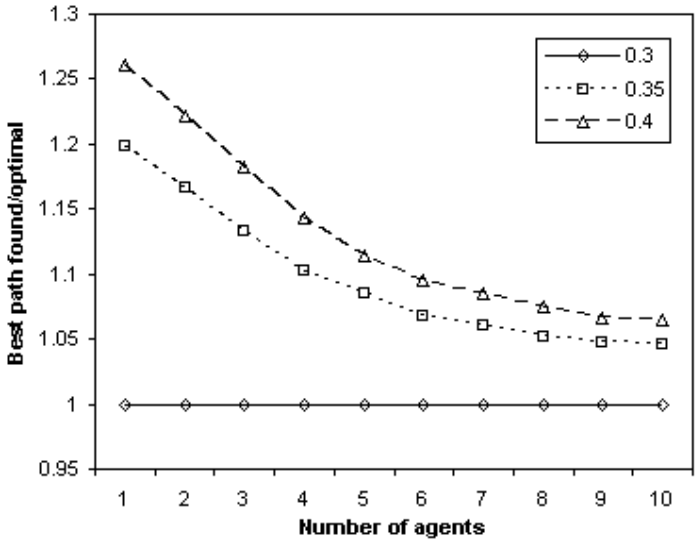

Fig. 3. DisNAV best agent. 1000 nodes

to the optimal. The combined result of figures 2 and 3 can be stated as follows. Using a minimal number of neighbors and using a large number of agents, the harder problems (low density graphs) are solved close to optimality.

Figure 4 presents a comparison between DisNAV and MAPHA*, the multi-agent physical A* algorithm of [Felner *et al.*, 2004] on a graph with 500 nodes and a density of 0.3. Note again, that MAPHA* actually activates a systematic global search and is guaranteed to return an optimal solution. The 2 curves correspond to the length of the path traveled by the first agent running DisNAV that reached the goal node, and the average path traveled by the agents performing MAPHA*. Recall from figure 3 that for a density of 0.3 the best path found by DisNAV is in practice the optimal path. As shown in figure 4, the first DisNAV agent that reaches the goal node, travels a shorter path than the average MAPHA* agent. This means that in practice in graphs with low densities it is more beneficial to use DisNAV agents than MAPHA* agents as they both find the optimal path in practice.

In another experiment the number of neighbors connected to each agent was varied. Figure 5 presents the length of the path traveled by the first DisNAV agent reaching the goal node, as a function of the number of agents. The four curves correspond to cases where there are no neighbors, when the number of neighboring agents is two and four and when all agents are connected to each other. The results show that adding neighbors will improve the total knowledge sharing and reduce the travel effort. However, the biggest jump is between the case when an agent has no neighbors and the case where each agent has two neighboring agents. We see a diminishing return when adding more neighbors for each agent.

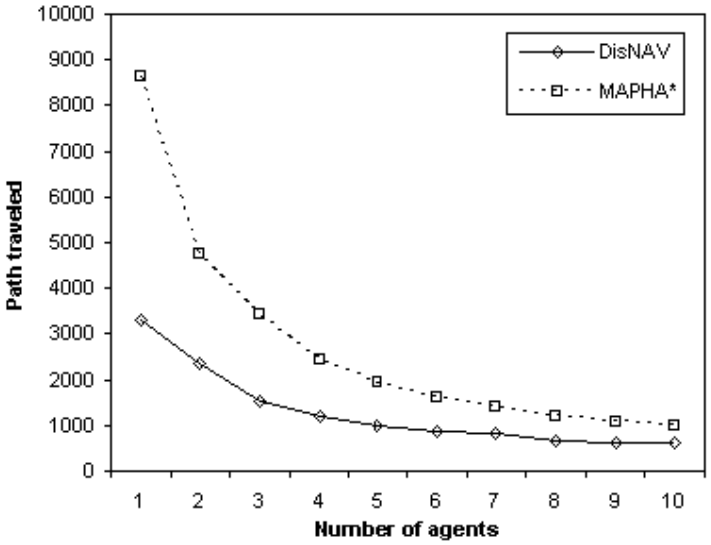

Fig. 4. DisNAV best agent, MAPHA* average. 500 nodes, density 0.3.

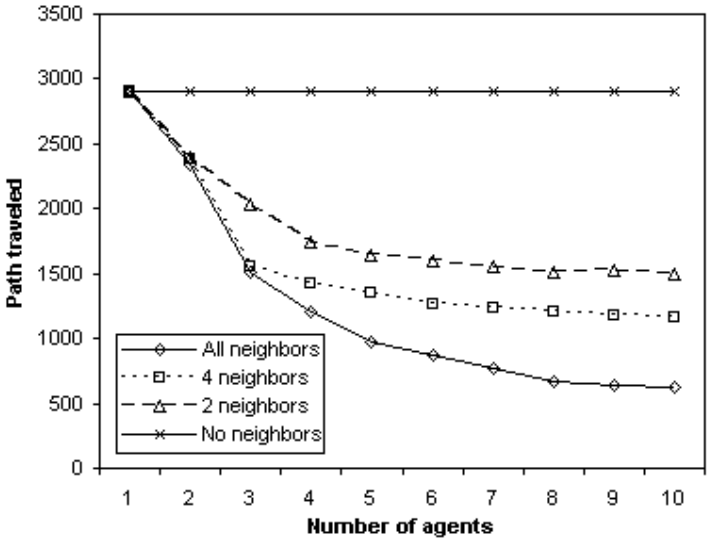

Fig. 5. DisNAV best agent. 500 nodes, density 0.3.

Figure 6 presents the average number of messages sent by an agent. The three curves correspond to different number of neighboring agents. It is clear from the results that

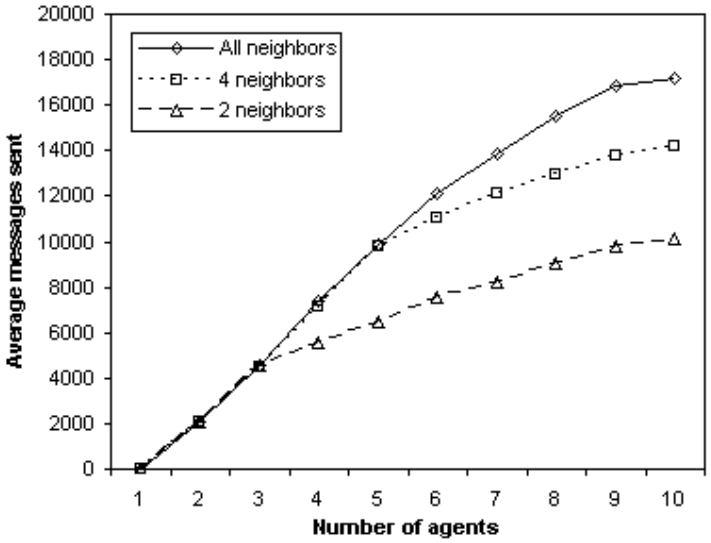

Fig. 6. Average message count. 500 nodes, density 0.3.

as the number of neighbors increases, the average number of messages sent by an agent increases as well.

5 Discussion

A multi-agent distributed algorithm for navigation in an unknown physical environment was presented. Our approach assumes that agents have limited communication abilities, and each of them is able to communicate only with a predefined set of agents. Furthermore, in our algorithm each agent is only navigating to the goal and we do not activate a systematic global search to find the optimal path. In contrast, MAPHA* [Felner *et al.*, 2004] activates a systematic global search and uses complete knowledge sharing between the agents. Therefore, the travel effort and communication overhead of our agents are much smaller than MAPHA*. However, even with the above limits, the experiments have demonstrated that our agents, using a simple distributed knowledge sharing, work together very efficiently. The path found by them in practice, was very close (and in many cases equal) to the optimal path. Thus, in practice, unless the optimal path is a hard requirement, our distributed algorithm is preferable.

Our experiments show that as the number of participating agents grows, the path traveled by the first agent arriving at the goal node is shorter. This benefit is more remarkable as the graph density decreases. In denser graphs, however, there is no benefit because even a single agent would travel a relatively short path.

When looking at the quality of the best path found, compared to the optimal path, the behavior is different. In dense graphs, a single agent would find a path which is a good approximation of the optimal path and adding more agents would not improve the

results. However, in sparse graphs, the path found by a single agent will be far from the optimal path and adding more agents will improve the results dramatically.

In many cases an agent that arrived at the goal node can prove that the best path known to it is optimal. This can be done by activating A* from the initial node to the goal. If all nodes generated by A* are included in the *KnownGraph* then the agent can prove that it has the optimal path. The experimental results show that in small, sparse graphs (e.g., 200 nodes and a density of 0.3) the agents always found the optimal path and in 40% of the cases they could even prove that it is optimal.

In order to simplify the definition of the distributed navigation problem, the present study uses a static neighborhood where neighbors of each agent are predefined. In principle there can be several dynamic definitions for neighborhoods. For example, a natural definition is the current set of all agents physically located in a predefined radius around it or the closest k agents. This, however, would require each agent to dynamically discover its current neighbors. Another issue in this context is agents connectivity. It is natural to require that agents remain in a connected communication network. Keeping connectivity while using dynamic neighborhoods is a complex issue and is left out of the scope of the present investigation. Future work will examine how does the communication network topology, the neighborhood radius and the connectivity affects the overall performance. Another interesting question is to examine the cases where agents are starting from different initial nodes and all agents are required to arrive at the goal node in order to complete the task.

References

[Bender *et al.*, 1998] Bender, M. A.; Fernandez, A.; Ron, D.; Sahai, A.; and Vadhan, S. P. 1998. The power of a pebble: Exploring and mapping directed graphs. In *Proc. ACM Symposium on the Theory of Computing*, 269–278.

[Burgard *et al.*, 2005] Burgard, W.; Moors, M.; Stachniss, C.; and Schneider, F. 2005. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*.

[Felner *et al.*, 2004] Felner, A.; Stern, R.; Ben-Yair, A.; Kraus, S.; and Netanyahu, N. 2004. PHA*: Finding the shortest path with a* in unknown physical environments. *JAIR* 21:631–679.

[Hart, Nilsson, & Raphael, 1968] Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SCC-4(2):100–107.

[Korf, 1990] Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(3):189–211.

[Lynch, 1997] Lynch, N. A. 1997. *Distributed Algorithms*. Morgan Kaufmann Series.

[Okabe, Boots, & Sugihara, 1992] Okabe, A.; Boots, B.; and Sugihara, K. 1992. *Spatial Tessellations, Concepts, and Applications of Voronoi Diagrams*. UK: Wiley, Chichester.