Utility-Based Multi-Agent System for Performing Repeated Navigation Tasks*

Ram Meshulam
Computer Science Dept.
Bar-Ilan University
Ramat-Gan
Israel
meshulr1@cs.biu.ac.il

Ariel Felner
Information System
Engineering Dept.
Ben-Gurion University
Be'er-Sheva
Israel
felner@bgu.ac.il

Sarit Kraus†
Computer Science Dept.
Bar-Ilan University
Ramat-Gan
Israel
sarit@cs.biu.ac.il

ABSTRACT

Suppose that a number of mobile agents need to travel back and forth between two locations in an unknown environment a given number of times. These agents need to find the right balance between exploration of the environment and performing the actual task via a known suboptimal path. Each agent should decide whether to follow the best known path or to devote its effort for further exploration of the graph so as to improve the path for future usage. We introduce a utility-based approach which chooses its next job such that the estimation of global utility is maximized. We compare this approach to a stochastic greedy approach which chooses its next job randomaly so as to increase the diversity of the known graph. We apply these approaches to different environments and to different communication paradigms. Experimental results show that an intelligent utility-based multi-agent system outperforms a stochastic greedy multi-agent system. In addition the utility-based approach was robust under inaccurate input and limitation of the communication abilities.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent Agents, Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

mobile agents, applications of autonomous agents & multi-agent systems

*This work was supported in part by NSF under grant no. IIS-0208608 and by ISF grant no. 8008.

†Sarit Kraus is also afiliated with UMIACS.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. INTRODUCTION

Intelligent agents often have to perform tasks when they only have incomplete knowledge. In such cases, an agent should learn about the structure of its environment to better accomplish its tasks. Such learning involves exploration, in which actions are chosen for the goal of increasing the agent's knowledge, as opposed to exploitation where actions are chosen which directly lead to accomplishing the agent's given task. Exploring the world and learning its structure may be performed either in a separate exploration phase a priori, before performing any tasks, or it may be interleaved with task performance.

Acting with incomplete knowledge occurs in many domains such as motion planning and computer networks. In this paper we focus on physical mobile agents. We address the following problem. Suppose that an agent receives a task to travel back and forth between two locations (s and g) in the environment in a given number of times. For example, there are many items to deliver from s to g which cannot be delivered in one trip due to load capacity.

There are many algorithms which guide mobile agents in unknown physical environments. They can be classified as full exploration algorithms [3, 5, 7] or as navigation algorithms [4, 8, 12, 10, 7]. Full exploration algorithms are used when the entire environment should be a priori mapped out [3]. Navigation algorithms are used when the agents need to reach a specific target location once [5, 4].

Only extreme cases of our problem belong to the two classes above. If the number of repeated tasks of our problem is very large (infinity), it would be most efficient to have an a priori exploration phase of the entire state space so that an agent can find the most efficient plan for its task. This would minimize the travel cost of the agents as the cost of the exploration phase will be amortized over the large number of repeated traveling. If however, we only need to reach the target once then any navigation algorithm will suffice.

Our problem in its general form is a continuum of both classes as the exploration degree depends on the total travel cost of the mobile agents and on the bounded number of times that the target should be reached. While there are many solutions to these two extreme cases, there has been

almost no work on solving problems where the number of repetitions of a particular task is bounded. Thus, we present a unique algorithm which maintains the correct balance between exploration and exploitation.

Argamon et al. [2, 1] addressed this problem for a single mobile agent. They showed that in such cases, it is more effective to interleave exploration of the environment during the actual performance of the task. This causes the agent to learn better ways to perform it over time. They provided an algorithm that chooses the best balance between exploration of unknown regions and exploitation of the best known path. This balance strongly depends on the given number of times that the task should be performed (and denoted as R).

In this paper we generalize their work for the case where there are a number of mobile agents. Each of these agents is capable of performing the task as well as exploring the environment. The balance problem here is more complicated since we must also consider actions and tasks of other agents. We present an efficient algorithm for distributing exploration and exploitation tasks among the different agents such that the total travel cost of the agents is minimized. This algorithm chooses tasks for the agents based on a global utility function. A utility value is given to the possible future tasks and agents are instructed to perform tasks with high utility. We compare this utility-based approach to a stochastic, greedy, task selection mechanism where agents randomaly choose whether to search for shortcuts (exploration) or to follow the best known path (exploitation).

We used two different communication paradigms. In the first there is full knowledge sharing between the agents and in the second agents exchange only partial information. Furthermore, we tested the algorithms in two different known models [2, 1, 4] of environments. Each assumes different initial input and different abilities of the agents. We compare the two task selection approaches both theoretically and experimentally under the four different ways to combine the two communication paradigms with the two models of environments. We provide strong experimental evidence to show that using a utility-based task-selection approach consistently outperforms a stochastic greedy task-selection approach in all four combinations. Furthermore, the utility-based approach was superior even when it used a more constrained communication paradigm than the stochastic greedy approach. The utility-based approach is robust over different communication paradigms and limitations and for any number of agents.

2. PROBLEM DESCRIPTION

Let $G = (V, E)$ be a weighted undirected graph. Suppose that an agent receives a task to travel back and forth between two vertices of G, s and g, a given number of times (denoted by R in this paper). If the agents know the entire structure of the graph then the optimal path between s and g could be calculated with ordinary algorithms such as Dijkstra's single-source shortest path algorithm or the A* algorithm. However, the agents only have partial knowledge about the graph. Thus, physical exploration of the graph is needed to learn about the existence of nodes and edges.

We would like to minimize the overall cost (defined below) of all the agents while R trips from s to g and back are performed by the agents. At each point of time, when an agent reaches a node, it must choose between walking on the best known path and exploring unknown edges which might reveal a shorter path. Note that revealing a new edge can create a shorter path even if the edge is not attached to any of the nodes of the known path.

2.1 Models of environments

We experimented with two models of environment each with a different set of assumptions concerning the initial capabilities of the agents as well as assumptions made about the environment. The first model of assumptions was also used by Aragamon, Kraus and Sina in [2] and is referred to as the AKS model in this paper. The second model of assumptions was used by Cucka, Netanyahu and Rosenfeld in [4] and is referred to as the CNR model in this paper.

2.1.1 AKS model

In the AKS model we assume that the entire set of nodes including their Euclidean coordinates are given as input. We also assume that some path between any two nodes is also known. The simplest way to achieve this is by providing a spanning tree of the graph as input. Thus, many edges of the graph are not known to the agents and the known paths might be much longer than the shortest path.

In this model, when an agent reaches a node n it has two options. The first option is to leave node n via an edge that is already known to the agent. We assume that the cost of traversing an existing edge (n, m) is given by the weight w of that edge. We assume that $w(n, m) \geq d(n, m)$ where $d(n, m)$ is the Euclidean distance between the two nodes. Another option is to try to find new edges connected to n. This is done by querying whether an edge exists between the node n to another node m. We define the cost of querying whether an edge (n, m) exists as follows:

$$query(e) = \begin{cases} w(n,m) & \text{If (n,m) exists} \\ d(n,m) & \text{otherwise} \end{cases}$$

After the query the agent is located in m if the edge exists and remains in n if it does not exist. The rational behind these settings is that querying might be that the agent actually tries to walk on the edge. If it succeeds, it finds itself at node m at a cost of $w(n, m)$. If it fails then it is back in n with a cost penalty of $d(n, m)$.

2.1.2 CNR model

In the CNR we apply a completely different set of assumptions. First, we assume that the input of the problem only contains the coordinates of the start and goal nodes. Other nodes as well as all the edges are not known in advance. In this model we also assume that once in a node an agent can sense the directions of outgoing edges without knowing which node they lead to.

In this model, when an agent reaches a node it has two options. The first option is to leave node n via an edge (n, m) where the destination node m is already known to

the agent. The other option is to explore an outgoing edge whose destination is unknown. Once the agent reaches the destination node m it learns about it and adds the existence of that node as well as the existence of that edge (n, m) to its knowledge databases. In both cases, after taking the action the agent is located in node m and the cost of the action is $w(n, m)$.

2.2 Communication models

There are many paradigms for communication in multi agent systems. The most trivial model is complete knowledge sharing where any new discovery of an agent is immediately shared with all 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.

In our work we have used two paradigms of communication namely full knowledge sharing (FKS) and partial knowledge sharing (PKS). Other paradigms will be a subject for future work.

2.2.1 Full knowledge sharing

In FKS we assume that any information gathered by one agent is known to all of the other agents at once. Furthermore, the intentions and the next task taken by each agent is also known. This framework can be obtained by using 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. Note that the assumption of knowing intentions of other agents is especially valid with the centralized supervisor model since the supervisor surely knows the next task of each of the agents. However, even for the broadcasting model we can assume that each agent broadcasts its intentions and its next task.

2.2.2 Partial knowledge sharing

In the PKS paradigm we assume that each agent is autonomous and makes its own decisions. In order to limit the broadcast rate we assumed that an agent can broadcast messages only under special circumstances and that only a small amount of knowledge can be broadcasted. Therefore, an agent only broadcasts the following two messages:

- The fact that R has been decremented when the goal node has been reached.
- The existence of the edge and its weight after an edge query, if the edge exists.

Note, that if an agent realizes that an edge does not exist then it does not broadcast this fact and this is not known by the other agents. Also, locations of the agents and tasks assigned to them are not shared.

2.3 Performance measurement

In the case of mobile agents, most of the effort is devoted on physically moving them in the environment. We want to minimize the total cost of moving the agents. While many calculations are computed (e.g., initializing and updating the distance matrix) we omit the complexity of the computation time as it is insignificant (and is accomplished in fractions of a second) compared to the time of physically moving the agents. We measure the performance of the algorithm by the following two cost functions:

- **Time elapsed -** The amount of time needed for the agents to complete the task.
- **Resource cost -** The sum of the costs associated with the different actions of the agents, i.e., walking and querying (for the AKS model).

Note that when adding more agents to the system the time elapsed may decrease since more agents are working in parallel. However, the total resource cost of the agents will not necessarily decrease as the same amount of work should be done.

Algorithm 1 Algorithms Framework

mainLoop($spanningTree$,$start$,$goal$,R)

1. knownGraph=spanningTree.
2. Initialize D and keep it updated at all times.
3. Initialize $missingEdges = \phi$
4. While $R > 0$ do
   (a) Update relevant data structures.
   (b) $e = nextEdge()$.
   (c) Traverse e.
   (d) If $goal$ is reached then $goal \leftrightarrow start$, $R = R - 1$

3. ALGORITHMS FOR THE AKS MODEL

The algorithms presented in this section are for the environment assumptions and possible actions of agents of the AKS model. In the next section, we describe the changes that are needed to apply these algorithms to the CNR model.

At first, we present a high level framework which is identical to the different approaches and communication paradigms. Algorithm 1 is activated by each agent autonomously. The basic principle of the algorithm is very simple. As long as the goal node has not been reached R times, every time an agent reaches a node n it updates relevant data structures and chooses which outgoing edge to traverse. The following data structures are used by each agent:

- **knownGraph:** includes the subgraph that is currently known to the agent. It is initially set to include the spanning tree.
- **D:** distance matrix between all pairs of nodes. This matrix is initialized according to the distances of the spanning tree provided.

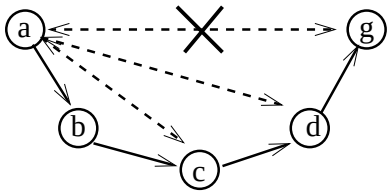

Figure 1: Possible shortcuts.

- **missingEdges:** this list contains pairs of nodes for which the agent is certain that edges between them do not exist.

Every time an agent queries about the existence of an edge and that edge exists, $KnownGraph$ and D are updated to reflect this new edge. Otherwise (the edge doesn't exist), $MissingEdges$ is updated. Once *goal* or *start* are reached, R is decreased and we flip *start* and *goal*. Note that once an agent reaches g it now needs to travel back to s. From the algorithmic point of view, this task is identical to the task of traveling from s to g. Thus R also includes the trips back from g to s. So if one needs to move 5 packages from s to g we set R to 10. The core of this algorithm is the $nextEdge()$ function which consults the agent about where to go next. This function is modified below for the various approaches and communication paradigms.

3.1 Algorithms for the FKS communication paradigm

Recall that in this paradigm agents share their entire knowledge. Thus every agent knows the location of other agents as well as their next task. Also, every discovery of existence or non existence (missing) of edges is shared by all the agents.

3.1.1 Stochastic greedy algorithm

Assume that the agent is located in node n and that the shortest known path to the goal is P. The agent can choose to travel along the known path P to the goal (exploitation) or to try to find shortcuts (exploration). The agent now consults the $nextEdge()$ in order to choose where to go next. This function for the stochastic greedy approach is defined as follows.

With probability p the agent looks for shortcuts. Otherwise, it follows the known path P. Looking for shortcuts is done by querying the existence of an edge (n, m) where $m \in P$ and (n, m) is neither in $KnwonGraph$ nor in $MissingEdges$. If there are many candidates for the node m the algorithm chooses the node that is closest to the goal node. If an agent chooses to query an edge, this fact is shared with other agents and they may not choose to query this edge again.

Consider the scenario presented in figure 1. The agent is located in node a. If the agent chooses to exploit the known path then it will go to node b. If it chooses to explore it will first query about the existence of the edge (a, d) (note that edge (a, g) is known to be a missing edge, or is already being queried by another agent). If it fails, (a, d) will be added to the $MissingEdges$ list and the agent will again choose whether to explore a new possible shortcut or to go via the best known path to node b. The idea behind this algorithm is to derive a simple but yet powerful mechanism for alternating between the possible two alternatives of exploring versus exploiting. Randomization is very effective as a method that introduces diversity and is known to be very powerful in local search and in solving combinatorial problems [9].

3.1.2 Utility-based algorithm

Here, we use a utility function to estimate the benefits of possible shortcuts and compare them to the utility of the known path. If all shortcuts have utilities which are lower than the utility of the known path, the agent follows the known path. Otherwise, the agent chooses to explore the shortcut with the highest utility. This utility function defined below was first presented by [2, 1] for a single agent. We generalize this idea to the multi-agent case.

The utility of the known path (without exploring any new edge) is $U(P) = cost(P) \times R$ where P is the known path, $cost(P)$ is the cost of traversing P and R is the number of times that the task still needs to be achieved. Therefore, the utility of exploring an edge e is

$$utility(e) = p(e) \times cost_s(e) + (1 - p) \times cost_f(e)$$

where

- e is an edge to be queried.
- $p(e)$ is the probability that e exists (p is equal to the density of the graph which is given in advance).
- Assuming that e does exist, $cost_s(e)$ is the new cost of completing the mission based on the fact that e exists ('s' stands for success). We estimate the cost of e.
- Assuming that e doesn't exist, $cost_f(e)$ is the new cost of completing the mission based on the fact that e doesn't exist ('f' stands for failure).

In other words, the utility of the query about e is the sum of the possible results of the query (existence or non-existence of e), weighted by the probability of each possibility to occur. Usually, a *utility* of an action is measured by considering the profit of the action and the cost of performing it. We want high profits at a low cost. Here, the profit is constant since the mission must be completed. Thus, in this paper when we talk about higher utility we actually mean low costs.

If there is a single agent working alone then the $NextEdge()$ function returns the task with the best utility among all possible tasks. This might be either a query about an unknown edge or following the best path.

When there are a number of agents the work should be distributed among them in an efficient manner. Recall, that in FKS, we assume that the agents share their locations and assigned tasks at all times. The $nexEdge()$ function for an agent working in a mutli-agent environment is defined as follows. It first identifies the best k shortcuts and store them

in a sorted list according to their utility where k is the number of agents. After deleting shortcuts which are assigned to other agents an agent chooses the shortcut to which it is the closest than all the other agents. If it is not the closest to any of the candidate edges, then:

- If the performance measure is the resource cost - the agent will remain idle until new knowledge changes the circumstances.
- If the performance measure is the time - the agent will go to the edge where it is second close etc,

The notion behind this difference is that in this case the agent cannot find a useful task. Therefore, if the performance measurement is the total resource cost it should remain idle. If however, the cost is the total time then it is better to choose any task than to remain idle. This observation was also noticed by [5, 6]. The idea to assign an agent to a nearby node was also used in MAPHA* [5] which activates A* in a real physical environment by a number of mobile agents.

3.2 Algorithms for PKS

As described above, in PKS an agent only broadcasts two types of messages, i.e., the existence of an edge and that R has been decremented. Missing edges, locations of agents and assigned tasks are not shared. We have observed that with this setting the communication rate is only 5% of the communication rate of the FKS paradigm because we do not broadcast about missing edges and tasks of other agents.

3.2.1 Stochastic greedy algorithm

The stochastic greedy algorithm for PKS is identical to the one presented for FKS. Here also an agent first chooses whether to explore a new edge with a probability of p. Exploration is conducted greedily as described for the PKS paradigm.

3.2.2 Utility-based algorithm

Suppose that we have k possible edges to explore. In the FKS paradigm above, these k edges would be distributed among all the agents for querying. The results of all these queries would also be shared among the agents. Thus, if a query failed, the fact that this edge does not exist is now known to all the other agents. Here, however, each agent acts autonomously. Agents do not know the location of other agents and are not informed about new missing edges. Thus, we face the problem that the same missing edge will be repeatedly queried by different agents.

In order to address this problem the selection algorithm employed by the agents should have the following main features:

- Every edge queried by the single agent version of the utility-based algorithm will be queried in a high probability by the multi-agent version of the algorithm.
- The number of cases where agents query edges that were already queried by other agents should be minimized.

We achieve these requirements as follows. First, the agent identifies the exploration edge e^* with the highest utility as described above for the FKS paradigm. Then, with a probability of p, the agent chooses to explore e^*. Otherwise, (with a probability of $(1-p)$) the agent treats this edge as if it does not exist, adds this edge to its own $missingEdges$ list and runs the nextEdge() function again. All the agents follow this stochastic approach.

Choosing the right p is very important. With a high value of p agents will tend to re-query edges that were queried by others. With a low value of p, however, agents might skip the querying of high-utility edges. To obtain a proper balance we introduce the following formula for choosing p. Let e be an edge, let q be the probability the edge e will be actually explored by at least one agent and let n be the number of agents. Note the relation between p, q and n is $q = 1 - (1-p)^n$. Extracting p from this reveals

$$p = 1 - \sqrt[n]{1-q}.$$

For example, if we have 8 agents and we would like to ensure that every edge will be explored at least once with probability of 90% - we set $p = 1 - \sqrt[8]{1-0.9} \approx 0.25$. This also means that each edge would be visited $n * p = 8 * 0.25 = 2$ times on average.

A stochastic distribution of tasks among agents was also presented in [11]. They generalized the best-first minimax search algorithm for the case of parallel processing. The best-first minimax algorithm is a variation of minimax search for game trees where the next node expanded is chosen according to an evaulation function. In order to parallelize the algorithm, each processor uses a probability function to choose the next node to expand. While they chose their probability according to a heuristic evaluation value attached to each node we provided the formula for p above which proved to be very effective in the experiments described below.

4. MODIFICATIONS FOR THE CNR MODEL

A number of changes to the algorithms should be made to modify them for the CNR model. First, recall that here exploration means traveling along an outgoing edge and reaching its destination. Also, since a spanning tree is not given, the agent(s) first need to navigate to find an initial path between the two given nodes. This can be done with any navigation algorithm. We used a multi-agent version of hill climbing when nodes are ordered by their utility. After an initial path is found the utility based agent uses the utility function to rank the potential shortcuts.

Another difference is the calculation of the utility function. In the AKS model, looking for a shortcut between two nodes n and m was done by querying whether the edge between them, (n, m), exists. Here, however, looking for a shortcut between n and m is done as follows. For each outgoing edge from n we calculate a utility value estimating the possible shortcut via this edge to m. Similar to the definitions above, given an outgoing edge e we define: $cost(e, n, m) = d(n, m) \times (c_1 + c_2 \times angle(l(n, m), e))$ where $angle(x, y)$ is the angle between the two lines x and y, and $l(n, m)$ is the straight line between n and m. We also estimate the exploration overhead cost which is the cost overhead of searching a path

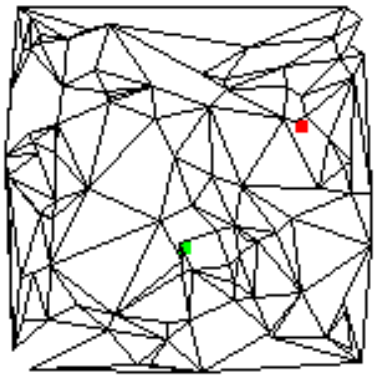

Figure 2: Delaunay graph of size 100 and density 0.05

between m and n: $explore(n,m) = c_3 \times d(n,m)$ We now define the utility function[1] as:

$$utility(e,n,m) = R \times cost(e,n,m) + explore(n,m)$$

Another modification to the CNR model is that in the PKS paradigm agents broadcast only new paths that decrease the length of the shortest known path.

5. EXPERIMENTAL RESULTS

We experimented with triangle graphs in which edges are generated based on the Delaunay triangulation of a random point set in the plane. These graphs are commonly used to simulate physical navigation tasks [5, 4, 2, 1]. We experimented with graphs with 100, 500, 800 and 1000 nodes. To simulate shortcuts (e.g., highways between distant cities in a roadmaps) we added random edges to these graphs until a desired density of the graph was reached. We used densities of 0.05 and 0.1. Figure 2 shows graph of size 100 and density 0.05. The dots on the graph are the start and goal points. Experiments with all these graphs confirmed the same tendencies provided below.

Figure 3 presents the elapsed time needed to complete the repeated task as a function of the number of agents for the different approaches on a graph with 100 nodes and a density of 0.05. Figure 4 shows the overall resource cost (defined in section 2.3) consumed by the agents. Both figures 3 and 4 refer to the AKS model. Each data point (in all our figures) is the average over 100 different runs. Unless otherwise stated, the number of repeated tasks was set to 100.

As can be expected, with all configurations the elapsed time decreases when more agents are added to the system. Also, we can clearly observe a diminishing return when adding more agents. This is not the case with the resource cost which tends to remain about the same (except for the stochastic greedy algorithm with PKS).

The results show that a utility-based approach clearly outperforms a stochastic greedy approach in both communica-

[1]We have used $c_1 = 2$,$c_2 = \frac{1}{180}$,$c_3 = 2.5$

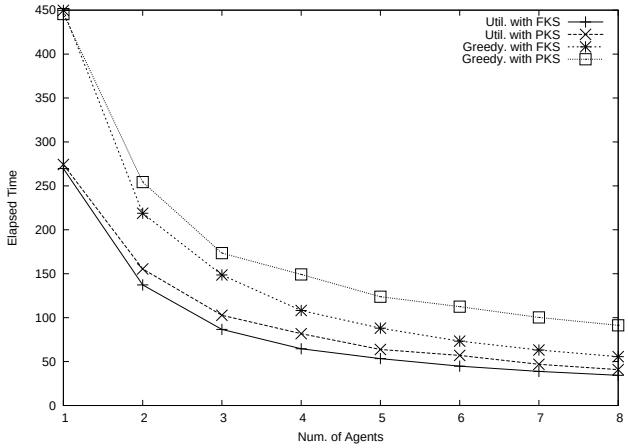

Figure 3: Time for graph with 100 nodes and a density of 0.05 for the AKS model

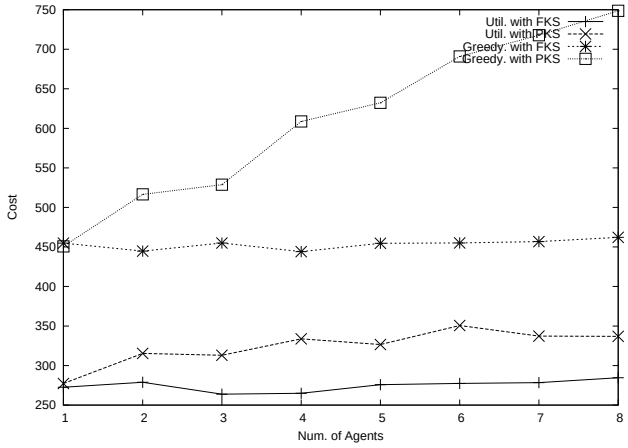

Figure 4: Cost for graph with 100 nodes and a density of 0.05 for the AKS model

Graph size	100			500			800			1000		
Agents number	4	6	8	4	6	8	4	6	8	4	6	8
Greedy for PKS	2.31	2.51	2.66	1.84	2.04	2.13	1.61	1.78	2.04	2.13	2.88	3.23
Greedy for FKS	1.68	1.64	1.62	1.53	1.57	1.67	1.45	1.46	1.52	1.74	1.94	2.09
Utility for PKS	1.27	1.27	1.19	1.08	1.11	1.12	1.09	1.10	1.11	1.12	1.15	1.16
Utility for FKS	1	1	1	1	1	1	1	1	1	1	1	1

Table 1: Time ratios over the utility-based approach for FKS for different graphs for the AKS model

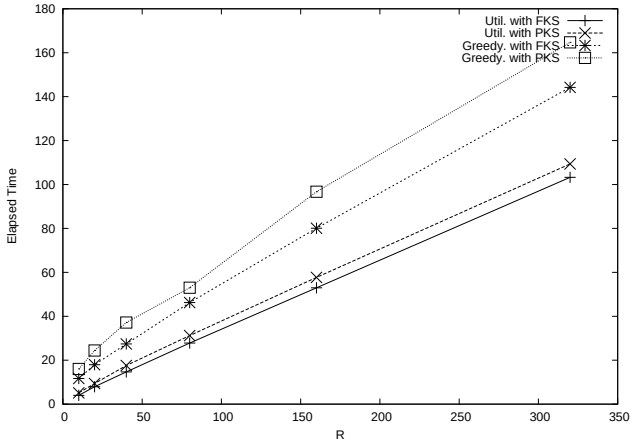

Figure 5: Elapsed time for increasing number of repeated tasks

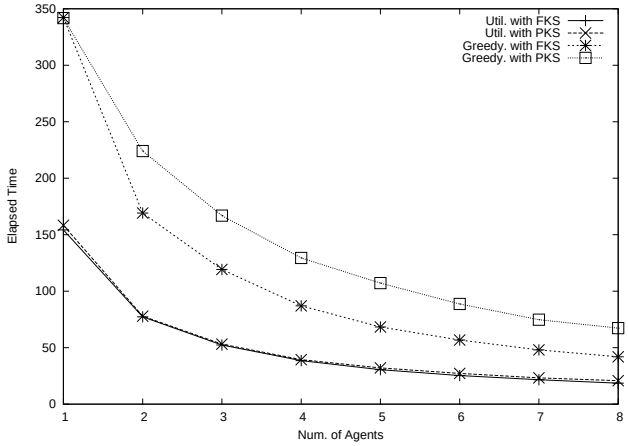

Figure 6: Time for graph with 500 nodes and a density of 0.05 for the CNR model

R	10	20	40	80
Greedy PKS	113.73	150.59	185.03	180.40
Greedy FKS	80.85	93.75	99.64	105.49
Utility PKS	22.16	42.00	56.11	63.90
Utility FKS	17.70	29.53	34.22	40.77

Table 2: Num of queries with various R values

tion paradigms. Furthermore, the utility-based approach for the PKS paradigm even outperformed the stochastic greedy approach for the FKS paradigm. Even though the broadcast rate was much lower for PKS, the utility-based approach needed no more than 20% more time than the same approach for FKS.

Table 1 shows the relative time performance ratio between the different approaches and the utility-based approach on the FKS paradigm (considered as 1 in the figure) on different graph sizes for the AKS model. This table confirms that the improvement rate of the utility-based approach is significant.

Figure 5 show the elapsed time of the 4 algorithms as a function of different number of repeated tasks which was varied from 10 to 320. Here again we can see the the utility-based approach outperformed the stochastic greedy approach.

Table 2 presents the number of queries that were performed by the different algorithms for graph of size 500 and with 5 agents. The table shows that the number of queries is increased with increasing R. The reason is that with larger R it is more beneficial to explore more regions. Note that while the utility-based performed smaller number of queries it managed to find shorter paths and thus completed the mission faster as presented in figures 4, 3 and 5.

These results show that the utility-based approach is robust under different communication paradigms and under different number of repeated tasks. On the other hand, the stochastic greedy approach is not robust for different communication limitations and its performance degrades by a significant factor when more limitations on communications are introduced when moving from FKS to PKS.

Figure 6 and 7 presents results for the CNR model. Figure 6 presents the elapsed time of the different approaches as a function of the number of agents on graph of size 500 and density of 0.05. Figure 7 presents the results for the resource cost consumed by the agents. The results are similar to these of the AKS model: The utility-based approach outperforms the stochastic greedy approach in both communication paradigms. As in AKS model, the utility-based approach for the PKS paradigm outperforms the stochastic greedy approach for the FKS paradigm.

6. DISCUSSION AND SUMMARY

We have shown the strength of the utility-based approach. The results that we obtained are conclusive for both the AKS and CNR environment models even though these models are completely different. The fact that this was true for both paradigms, both models and both performance measurements supports the effectiveness of the utility-based ap-

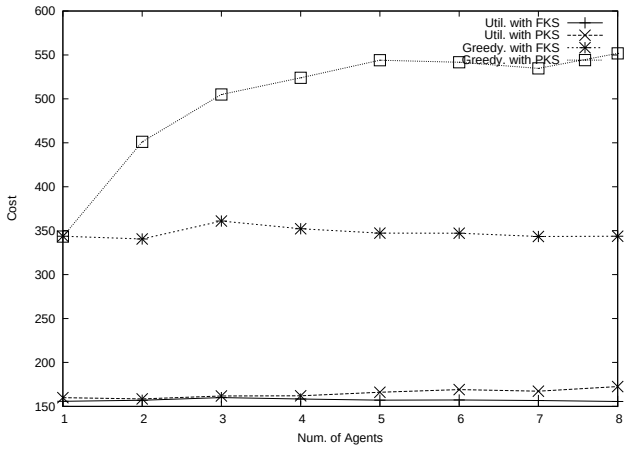

Figure 7: Cost for graph with 500 nodes and a density of 0.05 for the CNR model

proach.

Note that the results from both models show that if one can filter the best message types and only send such messages that inform about new existing knowledge then the performance of the algorithm with partial knowledge sharing will not significantly decrease even if the communication rate significantly drops. This is very encouraging as usually, communication abilities between agents seem to be very important.

We presented a multi-agent solution to the repeated-task problem. We compared a utility-based approach which chooses its next action in order to maximize a global utility with a stochastic greedy approach which decides about its next action according to a probability function. These algorithms were modified for the FKS and PKS communication paradigms. Our experimental results show that in both paradigms an intelligent utility-based multi-agent approach outperforms the stochastic greedy approach in both time and cost. Furthermore, we showed that the utility-based algorithm outperforms the stochastic greedy algorithm even when it has inferior communication and knowledge sharing paradigms. This implies that a utility-based approach is preferable in choosing between exploration and exploitations and in choosing which exploration step to perform.

Future work will continue in the following directions. Here we assumed that the environment is static and unchanged over time. The same ideas should be tested and modified to a dynamic environment where nodes and edges of the graph are not stable. A possible application could be ad-hoc networks which is of great interest in the research and industrial communities.

Moreover, many assumptions made on this paper should be challenged to make the algorithm useful to real world robots. For example, the algorithm should handle communication failures and limit the number of agents in one node at the same time. Another enhancement of the algorithm is to assign cost to communication between agents and embed communication cost in the utility function.

Another direction will be to modify the utility-based approach to other communication paradigms and to other problems that are solved by multi-agent systems. This might highlight the attributes and benefits of the utility-based approach.

7. REFERENCES

[1] S. Argamon-Engelson, S. Kraus, and S. Sina. Utility-based on-line exploration for repeated navigation in an embedded graph. *Artificial Intelligence*, 101(1-2):267–284, 1998.

[2] S. Argamon-Engelson, S. Kraus, and S. Sina. Interleaved versus *priori* exploration for repeated navigation in a partially-known graph. *IJPRAI*, 13(7):963–896, 1999.

[3] 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*, pages 269–278, May 1998.

[4] P. Cucka, N. S. Netanyahu, and A. Rosenfeld. Learning in navigation: Goal finding in graphs. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(5):429–446, 1996.

[5] A. Felner, R. Stern, A. Ben-Yair, S. Kraus, and N. Netanyahu. PHA*: Finding the shortest path with A* in unknown physical environments. *Journal of Artificial Intelligence Research*, 21:631–679, 2004.

[6] L. Finkelstein, S. Markovitch, and E. Rivlin. Optimal schedules for parallelizing anytime algorithms: the case of independent processes. In *Proc. AAAI-02*, pages 719–724, 2002.

[7] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comput. Surv.*, 24(3):219–291, 1992.

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

[9] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 2002.

[10] L. Shmoulian and E. Rimon. Roadmap-A*: An algorithm for minimizing travel effort in sensor based mobile robot navigation. In *Proc. ICRA*, pages 356–362, Leuven, Belgium, May 1998.

[11] Y. Shoham and S. Toledo. Parallel randomized best-first minimax search. *Artif. Intell.*, 137(1-2):165–196, 2002.

[12] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. ICRA*, pages 3310–3317, San Diego, CA, May 1994.