A Scalable, Distributed Algorithm for Efficient Task Allocation

A Scalable, Distributed Algorithm for Efficient Task Allocation

Pedro V. Sander
Harvard University
33 Oxford Street
Cambridge, MA, USA
pvs@eecs.harvard.edu

Denis Peleshchuk
Harvard University
33 Oxford Street
Cambridge, MA, USA
dpeleshc@fas.harvard.edu

Barbara J. Grosz
Harvard University
33 Oxford Street
Cambridge, MA, USA
grosz@eecs.harvard.edu

ABSTRACT

We present a distributed algorithm for task allocation in multi-agent systems for settings in which agents and tasks are geographically dispersed in two-dimensional space. We describe a method that enables agents to determine individually how to move so that they are, as a group, efficiently assigned to tasks. The method comprises two algorithms and is especially useful in environments with very large numbers of agent and task nodes. One algorithm adapts computational geometry techniques to determine adjacency information for the agent nodes given the geographical positions of agents and tasks. This adjacency information is used to determine the visible nodes that are most relevant to an agent's decision making process and to eliminate those that it should not consider. The second algorithm uses local heuristics based solely on an agent's adjacent nodes to determine its course of action. This method yields improved task allocations compared to previous algorithms proposed for similar environments. We also present a modification to the second algorithm that improves performance in environments in which multiple agents are required to complete a single task.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems.

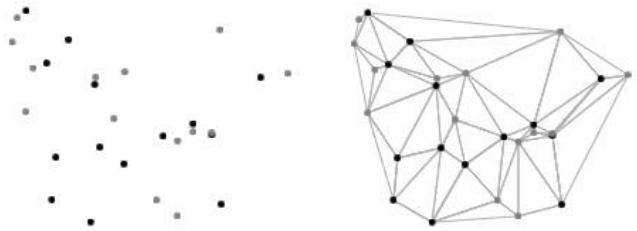
General Terms

Algorithms, Experimentation.

1. INTRODUCTION

Assigning agents to tasks in a non-centralized environment is a fundamental problem in multi-agent systems (MAS). The problem becomes even more challenging in settings in which a very large number of agents and tasks are spread in space, making communication among agents difficult or costly.

We propose an algorithm for task allocation based on computational geometry techniques. The approach is applicable to domains in which agents' and tasks' geographical positions are known. In our experiments, the tasks are stationary, but agents are allowed to move. Agents are aware of other agents and tasks only within a proximity radius, which we refer to as the agent's *circle of visibility*. The objective is to maximize the number of fulfilled tasks. We do not consider other costs associated with the agents, such as distance traveled or awake time.



a) Agents (gray), tasks (black) **b) Delaunay triangulation**

Figure 1: Agents and tasks as vertices on the 2D plane with the Delaunay triangulation shown on the right.

This algorithm is useful in numerous domains, including the package delivery system of Shehory *et al.* [1998]; a taxi company, where the drivers are the agents and the passengers are the tasks; and the allocation of police forces with no central control. In this last case, it is critical to have a good distribution of agents (policemen) because a new task (a crime) can appear anywhere.

The algorithm is fully distributed. Each agent determines its course of action with the objective of maximizing its own number of fulfilled tasks. To reduce communication costs, each agent bases its decision solely on a small set of adjacent task and agent nodes. The expected number of adjacent nodes remains low even if the density of agents and tasks is very high. Furthermore, the adjacent nodes are evenly distributed in all directions. For instance, it is undesirable to have all adjacent nodes of an agent be in the east direction, as that could lead to missing an opportunity to reach a nearby task to the west. Picking the n closest nodes could cause an imbalance. The algorithm computes a planar triangulation of the nodes at each time-step (Figure 1) and determine the adjacency of the nodes based on a subset of the edges of the triangulation.

Section 2 presents an algorithm to efficiently compute a local set of neighbors adhering to the properties outlined above. This set provides each agent with a small set of desirable neighbors on which to base its decisions.

Section 3 presents an algorithm for an agent to make decisions based solely on its adjacent nodes, rather than on all the nodes within its circle of visibility. This property of the algorithm minimizes the amount of communication and computation required by the agents. The adjacent nodes also provide enough information to allow the agents to spread themselves out evenly in space. Since all computations are done locally by individual agents, the algorithm scales very well.

AAMAS'02, July 15-19, 2002, Bologna, Italy.

The basic algorithm described in Section 3 is applicable to tasks that can be completed by a single agent. In Section 4 we describe a modification to the algorithm that is suitable for environments with tasks that require the simultaneous presence of multiple agents. The algorithm uses adjacency information to propagate data about tasks to farther agents, as needed.

In Section 5 we describe the implementation and the results of our experiments. We then place our system in the context of related work in Section 6. Finally, we conclude and propose some directions for future work.

2. PLANAR TRIANGULATION

This section describes an algorithm that uses a planar triangulation to allow each agent to identify and focus its task deliberation on a small set of adjacent nodes. Agents only consider these adjacent nodes when making their decisions. As a result, the amount of deliberation and computation is significantly reduced without compromising solution quality.

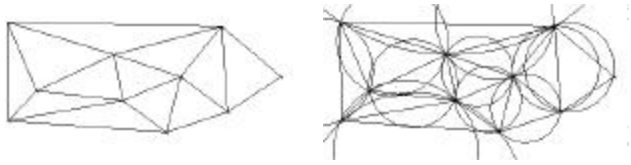
The planar triangulation problem seeks to join a set of points in the plane by nonintersecting lines, such that every region within the convex-hull of the set of points is a triangle. As a result, a planar triangulation yields a connected graph with generally short edges. A triangulation of the points in Figure 1a is given in Figure 1b. Planar triangulations is a well studied area within computational geometry and widely used in computer science. Applications for planar triangulations range from surface interpolation in graphics to routing algorithms in networking.

Each agent computes its own triangulation locally within its circle of visibility. We require agents to generate triangulations that are consistent with one another.

A set of points may be triangulated in many different ways. Our algorithm is based on the Delaunay triangulation [Delaunay, 1934]. A triangulation is considered to be Delaunay if every triangle's circumcircle[1] does not enclose any other point in the triangulation (Figure 2). The Delaunay triangulation maximizes the minimum angle of all triangles, so it has the desirable property that points are only adjacent to a generally small set of close neighbors in all directions. The circled node to the right, has many more close neighbors to the east, yet its adjacent neighbors are evenly spread in all directions.

Because agents are limited by their circles of visibility, each agent computes a *local* Delaunay triangulation. The edges of the local Delaunay triangulation form a superset of the edges of the global Delaunay triangulation that are inside the agent's circle of visibility. Some triangles in the local triangulation have large circumcircles, and in many cases these circumcircles might span outside the agent's circle of visibility. In such cases the agent will not be able to determine whether a given edge of the local triangulation is also in the global triangulation. Furthermore, the circumcircle test is not consistent between pairs of agents. If agent *A* has agent *B* as an adjacent node, agent *B* will not necessarily have agent *A* as an adjacent node; The circumcircles associated

[1] A triangle's circumcircle is the *unique* circle that passes through all of the triangle's vertices.



a) Delaunay triangulation **b) Empty circumcircles**

Figure 2: The Delaunay triangulation has the property that every triangle's circumcircle does not enclose any points.

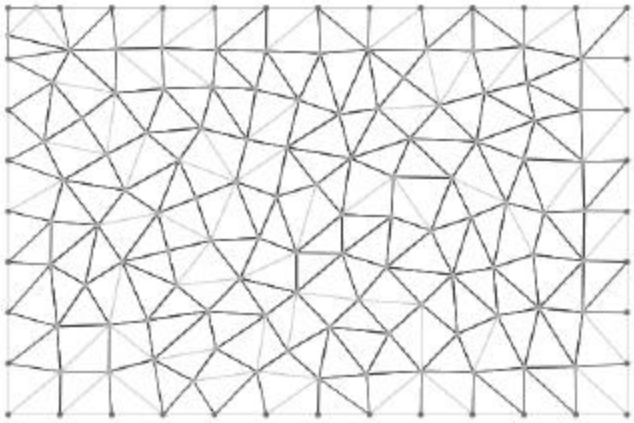


Figure 3: Edges of the Delaunay triangulation. Edges in dark gray passed the edge test, while edges in light gray failed.

with the edge might lie within the circle of visibility of *B*, but they might not lie within the circle of visibility of *A*.

To address this consistency problem, instead of using the circumcircle test, we perform the following simple test on an edge: An edge is in the triangulation, if the circle passing through both of its endpoints and with diameter equal to its length does not contain any other points (as shown in the example to the right). This simple edge test is stricter than the triangle circumcircle test and yields a subset of the edges of the Delaunay triangulation.

In practice, this method finds a high percentage of the Delaunay edges if agents are well distributed in space. As shown in Figure 3, typically only one or two Delaunay edges adjacent to each node failed the test. Every visible node can be tested for adjacency, since for all the visible nodes, their respective edges' circles will be completely inside the agent's circle of visibility. Furthermore, this method is consistent among different agents since these circles lie completely within the circle of visibility of both agents.

The final algorithm for an agent *A* to compute its neighbors is:

- Compute the Delaunay triangulation of nodes visible from *A*, using a standard $O(n \log n)$ algorithm [Preparata and Shamos, 1985].
- For every edge adjacent to *A*, check whether the edge passes the stricter edge test. The edge only has to be checked against the two nodes opposite the edge in the two adjacent Delaunay triangles. If the edge passes the test, then we keep it, otherwise we discard it.

This Delaunay triangulation step can be considered a black box by the agent. Adjacency can be computed very efficiently, and may

even be done in hardware. Once adjacency is established, agents can now use either the single-agent task allocation algorithm of Section 3 or the multi-agent one of Section 4.

3. BASIC ALGORITHM

The algorithm in Section 2 computes for each agent a set of adjacent nodes. This section gives an algorithm that uses adjacency information to determine the direction an agent should move. The objective is to move toward tasks and away from other agents. If one or more task nodes are adjacent to the agent, it will move in the direction of the nearest one in an attempt to reach it and fulfill it. If the agent has no adjacent task nodes, then it will repel its adjacent agent nodes to better cover empty spaces. To prevent the agent from being too close to a peer, the agent should repel nearby agents more than farther ones. Hence the repelling force should be inversely proportional to distance. We set it to be d^{-k}, where d is the distance, and k is a specified constant. More formally, given the adjacent agents j, the agent moves in the direction given by

$$\frac{\sum_j \left(\frac{v(j)}{\|v(j)\|} \cdot \|v(j)\|^{-k} \right)}{\sum_j \|v(j)\|^{-k}} = \frac{\sum_j \left(v(j) \cdot \|v(j)\|^{-(k+1)} \right)}{\sum_j \|v(j)\|^{-k}},$$

where $v(j)$ is the vector from the geographical coordinates of the adjacent agent j to the current agent's coordinates.

We ran experiments to determine the best value for the k parameter, and the results for using different values of k were all quite similar, as long as $k>0$. We decided to use $k=2$, as it gave slightly better results.

With a sufficiently low number of agents, the agents will repel one another and end up at the boundaries. To prevent this, we introduce "phantom nodes", imaginary nodes evenly spaced along the boundaries. In the absence of nearby tasks, agents will also repel the phantom nodes. With this addition, agents "bounce" off the boundary. The number of phantom nodes is set to $\sqrt{n}$ at each boundary, where n is the number of agents.

Tasks only attract adjacent agents. Agents that are not adjacent to a particular task have a very low probability of being able to reach that task before other agents do. This algorithm ensures that non-adjacent tasks do not influence where an agent moves. For instance, if there is only one task to be satisfied, the adjacent agents will all attempt to reach the task, while other agents will not even try. Instead, they will try to evenly spread themselves in space in the expectation that another task might appear in their vicinity.

The algorithm effectively shrinks and grows the agent's circle of visibility based on the density of nodes. If there are a lot of nodes, the agents will only pursue nearby tasks, while if there are very few nodes, the agents can try to reach tasks that are farther away.

Figure 4 shows two screenshots of our system with significantly more agents than tasks. Using the simple algorithm outlined above only nodes that are adjacent to tasks are attempting to reach them; the others remain evenly spaced.

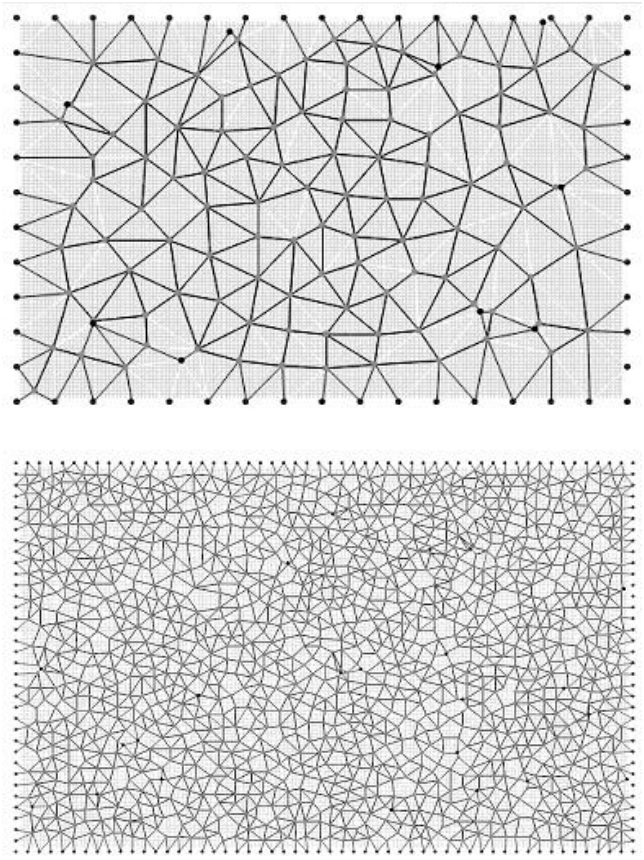


Figure 4: Two screenshots of our system. The agents (gray points) are well spaced. The tasks (dark interior points) only attract adjacent agents. Agents repel other agents and phantom nodes (dark boundary points) in the absence of adjacent tasks.

4. WEIGHT SYSTEM

In this section we describe an extension to the algorithm that addresses scenarios in which tasks require multiple agents in order to be completed. In this algorithm, each task is given a weight equal to the number of agents that are needed to complete it. Figure 5 shows a screenshot of our program using the weight system.

With only a slight modification to the basic framework so that a task node with weight w does not disappear until w agents reach it, the basic algorithm performs well for low weight values since, on average, a task is directly adjacent to enough agents. When the weight is increased to values greater than 5, the performance degrades substantially as only immediate neighbors are aware of a task and move directly towards it. Agents a little farther away have no information about the task until direct neighbors reach it and are removed from the triangulation.

As a solution to this problem, we propose a weight propagation mechanism based on local communication between adjacent agents. The triangulation provides an ideal basis for this approach as it introduces a connected graph while still limiting the number of adjacent neighbors and thus the amount of communication.

The algorithm works as follows. Initially, all agent nodes are assigned a weight of 0 and all task nodes are assigned a weight equal to the number of agents needed to satisfy them. If an agent is only adjacent to zero-weight nodes, it moves away from its neighbors to achieve a more even distribution in space, as in Section 3. Otherwise, it picks the neighboring node with the highest weight, which we will now refer to as the parent node. In case of a tie, it picks the closest node. If the parent node is a task, it moves towards it. If it is an agent, it gets the destination task information from that agent and moves towards that same task. The weight of the agent is set to $w(n) \cdot 2^{-k}$, where $w(n)$ is the weight of the parent and k is a constant that regulates the rate of weight propagation. If the weight becomes less than 1, it is set to 0. Thus, when all task weights are 1, this algorithm behaves identically to the algorithm described in Section 3.

The example in Figure 6 illustrates the algorithm with the rate of weight propagation set to 1. In this scenario, there are two tasks, one with a weight of 8 and the other with a weight of 1. The weights of tasks are propagated through agents and set to the values shown. As a result, only the agent on the lower left will move to the task with a weight of 1. The others will be attracted to the task with a weight of 8. This example demonstrates the advantages of weight propagation as farther agents become aware of the task without waiting until agents that are in between reach it and are removed from the triangulation. Furthermore, with weight propagation, agents can become aware of a task even if the task is outside of the their circles of visibility.

When using this algorithm, it is important to address the trade-off between responsiveness and efficiency. We want the system to be responsive so that when a new task appears, enough agents are attracted to it. However we also want it to be efficient and avoid redundancy, which is created when too many agents move into the same area, potentially increasing response time for tasks appearing somewhere else. To regulate this behavior, the maximum number of levels of propagation as well as the k parameter can be adjusted for a particular task-to-agent ratio.

Since this system involves only very minimal coordination between agents, if the ratio of tasks to agents is very high, agents could move to different tasks and wait there for help indefinitely. Two different approaches may be taken to address this problem. The weight of a task node can be increased as agents get there to effectively indicate that it becomes more important to satisfy that task now because some agents are waiting there and cannot do any other useful work. An alternative approach, which can be used independently or in conjunction with the first one, is to introduce a time-out value to represent the maximum amount of time an agent waits for help. After that time elapses, the agent will move away from the task. This second approach was implemented in our system.

The issues described above are important, and they raise the question of whether more comprehensive cooperation mechanisms should be introduced into the system. With no communication and computation constraints, it would be possible to avoid conflicting and redundant efforts and the system would be perfectly coordinated [Malone, 1987]. Under restrictions of practical distributed systems, however, this is impossible to achieve. When designing such systems, it is important to ensure that agents spend most of their time solving the domain level problems for which they were built, rather than in communication and coordination activities [Jennings, 1996].

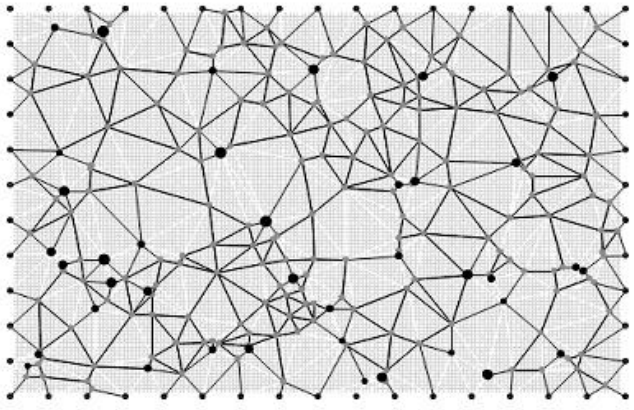


Figure 5: Screenshot with varying weight tasks. Larger points represent tasks with higher weights.

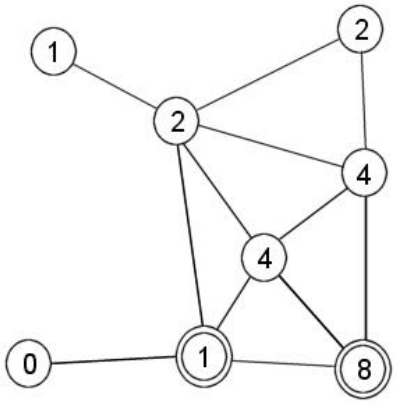


Figure 6: Weight propagation of task nodes (double-circled) through agent nodes (single-circled).

5. IMPLEMENTATION AND RESULTS

We implemented a system using the methods outlined above. For computing Delaunay triangulations, we used the Qhull software library [Barber and Huhdanpaa]. To compare our results to the physics-based approach of Shehory *et al.* [1998], we use settings similar to theirs. The domain is a 30-by-20km city and the streets are arranged in a square lattice, such that each city block is 200m long (150 streets by 100 streets). Like in Shehory *et al.*'s freight delivery system, after each agent arrives at a task node, it must deliver the freight to some random city destination. Once the freight is delivered, the agent reappears at the delivery point and resumes searching for tasks. Even though we used the freight delivery example, the results are indicative of how the system would perform in other similar settings, such as the police force or taxi company. To compare our results to those of Shehory *et al.* [1998], we used settings similar to theirs. We performed our runs with 250 agents, 1,200 initial tasks, 600 new tasks per hour, and agents traveling at 50km/h.

5.1 Equilibrium

The first experiment, shows how the system behaves during the initial stages until it reaches equilibrium. We performed two runs. In the first run, we set the radius of the circle of visibility to 0.5km, while in the second we set it to 2km. The results of the

experiment are shown in Figure 7. *Searching agents* represents the number of agents that are currently looking for a task. The *agent searching time* is how long an agent takes to find a task on average. *Waiting tasks* represents the number of tasks that are waiting for an agent to arrive. The *task waiting time* is how long a task waits for an agent on average. The *task fulfillment time* is the sum of the waiting time and the delivery time (the time that takes to go to a random city destination on average). The delivery time is only dependent on velocity, and the velocity remains unchanged in this experiment. As a result the fulfillment time graph is just a shifted waiting time graph, and is not shown here. It is interesting that the number of searching agents oscillates quite significantly. They do however satisfy all tasks, and keep the task waiting time very low.

Both runs reached equilibrium, in that the agents were able to handle all incoming tasks and keep the number of waiting tasks relatively steady. The number of agents that are searching for tasks and the average time for an agent to find a task are very similar, since in both cases the agents are getting the same amount of work done. However, the number of waiting tasks and the task waiting time is significantly higher in the 0.5km visibility setting. In this setting the agents cannot see very far away. As a result the agents could not immediately find tasks, so the system reached equilibrium with a higher number of waiting tasks.

5.2 Circle of visibility

We performed an experiment varying the radius of the circle of visibility from 0.4 to 5 km. The results after 70 hours of simulated time are shown in Figure 8. Again, all the systems reached equilibrium, so the number of searching agents and the searching time are very similar for all cases. The number of waiting tasks and the task waiting time decrease as the radius increases. We were concerned that the system would require a very large radius, but these results indicate that with the default parameters and 250 agents or more, using a radius greater than 2 yields no significant improvement. At that radius there are enough agents so that they can almost always see their closest neighbors in all directions. We also tested the algorithm on cases in which different agents have circles of visibility with different radii, and the system performed equally well.

5.3 Number of agents and velocity

We performed tests with default settings and a varying number of agents and velocity. We set the circle of visibility to 2km. The length of each run was 70 hours of simulated time. Figure 9 contains the same four standard measurements of the prior experiments, while Figure 10 shows the task fulfillment time. For the prior two experiments, the fulfillment time graph is a shifted waiting time graph, but since we use different velocities in this experiment, the delivery time varies.

As Figure 10 shows, as the number of agents increases, the number of searching agents and the agent searching time both increase, while the number of waiting tasks and the waiting time both decrease. Once the number of agents falls below a threshold, the agents cannot satisfy the tasks at the current incoming rate, and the system does not reach equilibrium. Clearly, the higher the velocity, the lower the threshold. In all cases where equilibrium is reached, the same number of tasks are satisfied per hour. By increasing the velocity, agents reach tasks faster, but there are more agents looking for tasks.

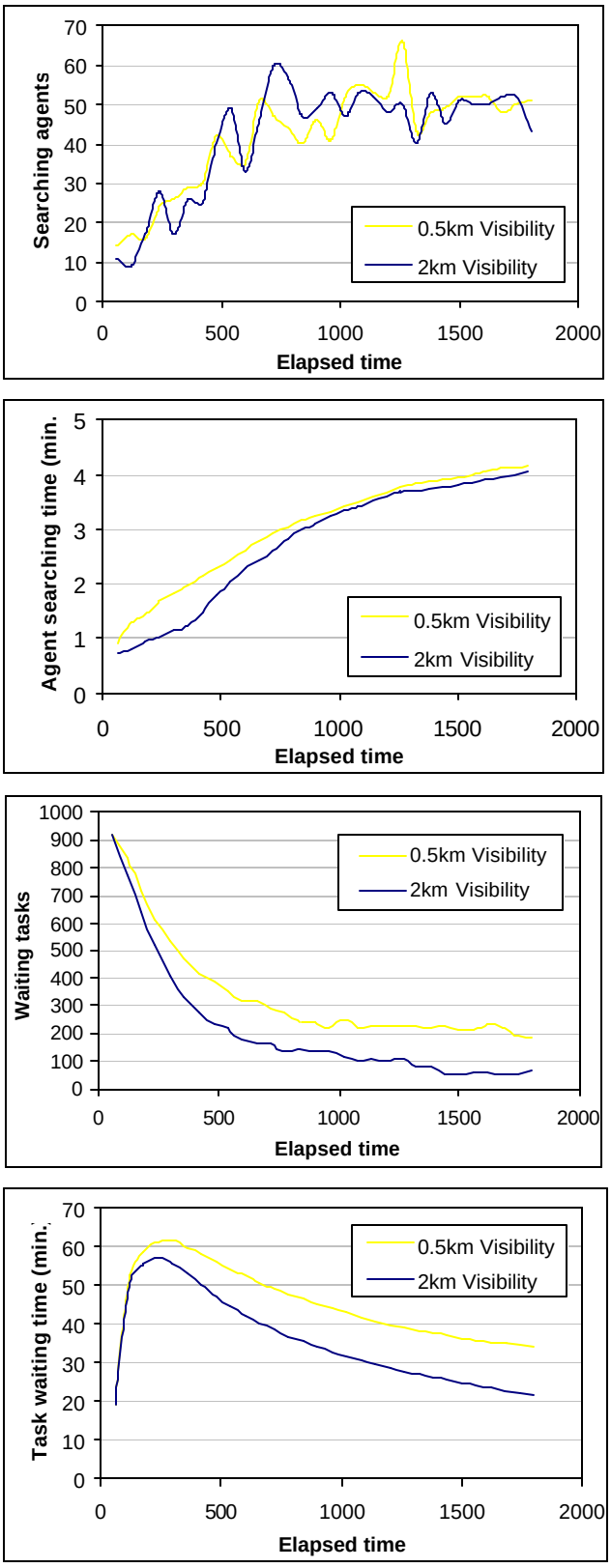
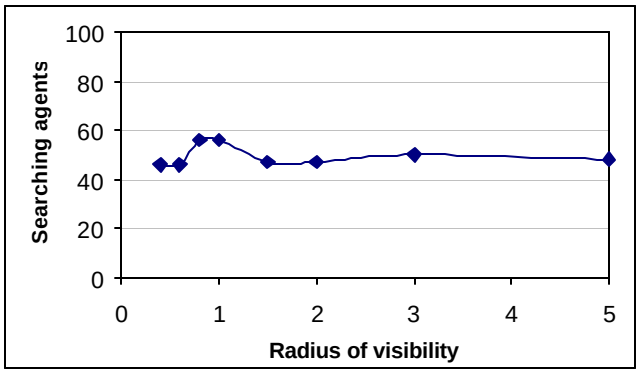
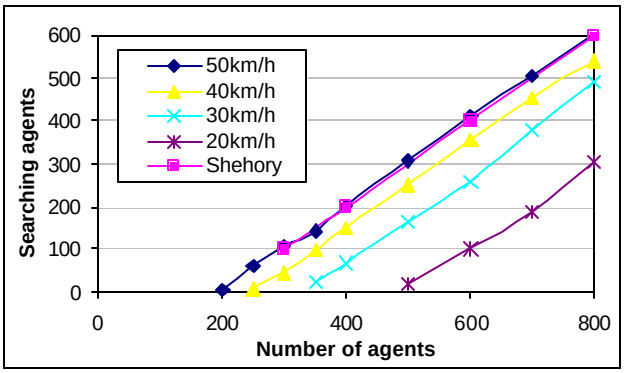
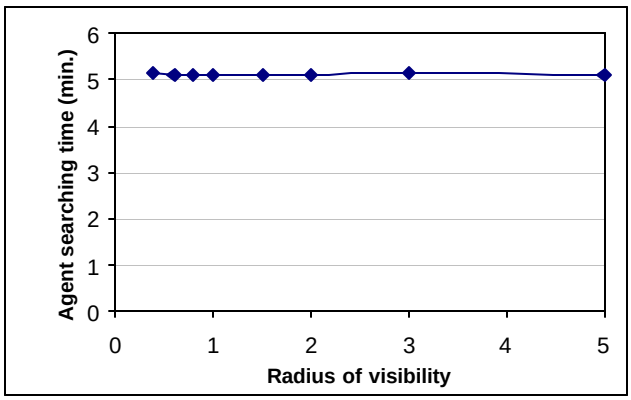
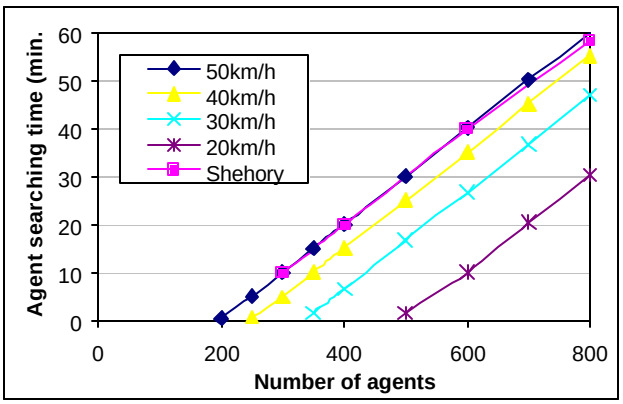


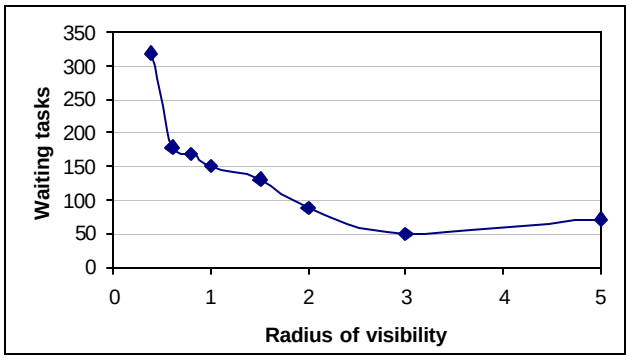
Figure 7: Results of two runs of our system with varying circles of visibility.

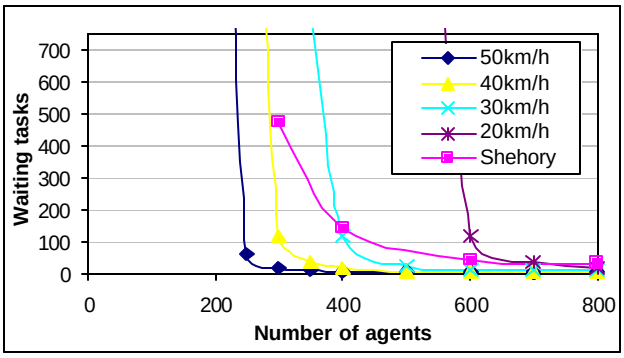


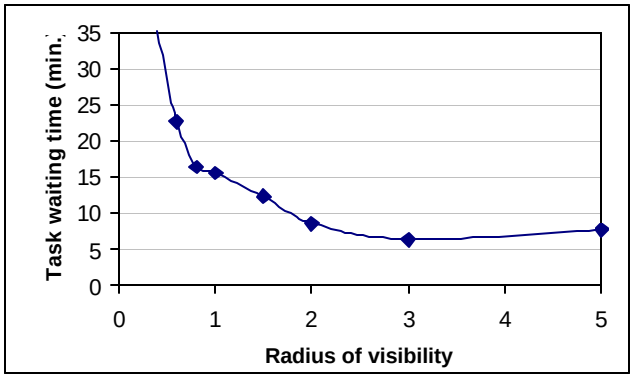












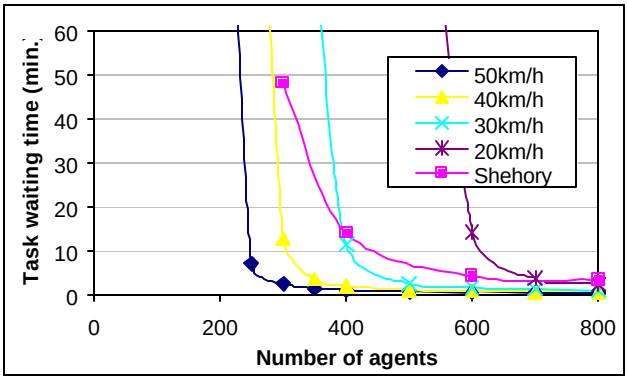


Figure 8: Results of our system as a function of the radius of the circle of visibility.

Figure 9: Results of our system varying the number of agents and the velocity. Results from Shehory *et al.* with a speed limit of 50km/h are also shown for comparison.

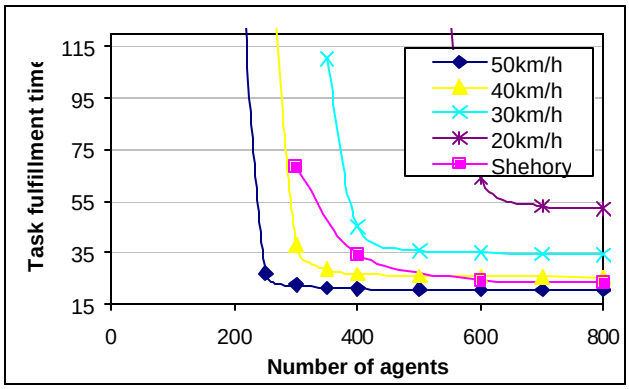


Figure 10: Average task fulfillment time of our system varying the number of agents and the velocity. Results from Shehory *et al.* with a speed limit of 50km/h are also shown for comparison.

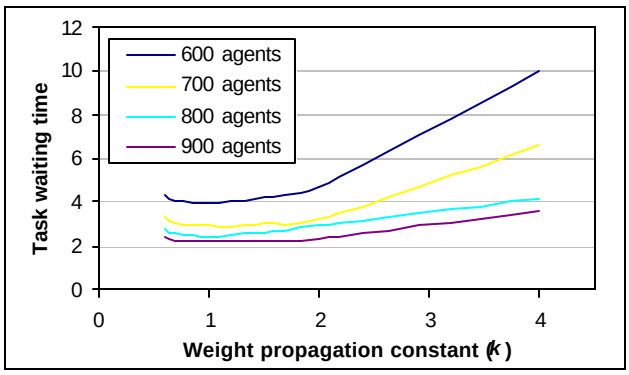


Figure 11: Results varying the weight propagation constant for different agent:task ratios.

These results demonstrate that our algorithm outperforms the system of Shehory *et al.* Shehory *et al.*'s experiments were run with a maximum velocity of 50km/h. As shown in Figure 10, our system fulfills tasks in less than half an hour using 250 agents, while their system requires approximately 600 agents to fulfill the tasks in the same amount of time. Shehory *et al.* use a city grid in which 10% of the city streets are closed, while we use a full grid. Examining their delivery time (fulfillment time minus waiting time), we noticed that they do not appear to be paying a significant penalty for not having a completely full grid, so we omitted a partial grid for our experiments. The agents in our system always travel at the maximum velocity when going towards a task node. In Shehory *et al.*'s system, even when the agent is approaching a task node it repels other agents. Furthermore, if the agent is farther from a task it will approach it at a lower speed. It will only reach the speed limit when the attraction force becomes large enough. As a result, there is a significant reduction in average velocity, causing an increase in the waiting time. Note that, with 400 agents, our system with a velocity of 30km/h has a comparable waiting time as theirs with a maximum velocity of 50km/h.

5.4 Weight system

We experimented to find the optimal weight propagation constant k for different task-agent ratios. For the experiment, we used 150 tasks with weights ranging from 6 to 10 while varying the number of agents from 600 to 900 and k from 0.6 to 4. Note that since the maximum task weight is 10, using k=4 is equivalent to using no weight propagation at all since the weight of agents immediately adjacent to the task is $10/2^4$, which is less than 1 and thus set to 0. Figure 11 demonstrates that there is no substantial difference in performance for k values ranging from 0.6 to 2. For values of k that are greater than 2, the system slows down significantly for a lower number of agents, but systems with a higher number of agents still perform well since, on average, more agents are closer to the tasks because of the higher density of agents. The optimal value of k decreases as number of agent increases since systems with a higher number of agents can afford more redundancy (lower values of k mean that the weight is going to be decreased at a slower rate and thus more agents will be attracted to a task).

6. RELATED WORK

Research in distributed systems has developed many algorithms that can be done locally in a distributed network. Locality is defined as a limit on time or distance, which is independent of the size of the network. Since the network has graph structure, many of these algorithms arise from graph theory. In such settings, many distributed graph theory algorithms attempt to find global solutions via local communication [Naor and Stockmeyer, 1993]. In contrast, our algorithm is purely local and does not require or attempt to find the global maximum.

Computational geometry techniques similar to the one presented in this paper have been used to solve problems in a wide range of areas where geographical position is important. Voronoi diagrams – the dual of Delaunay triangulations – have been used to place supermarkets so that they are evenly spaced around town. In the Networking field, several routing algorithms also make use of Voronoi diagrams and Delaunay triangulations. Meguerdichian, *et al.* [2001] uses Voronoi diagrams for optimal coverage calculation in wireless networks. Gao *et al.* [2001] propose a new routing graph for mobile ad hoc networks based on Delaunay triangulations.

Traditional AI research on task allocation has concentrated either on negotiation or on market strategies such as contracts [Sandholm and Lesser, 1995] or auctions that require substantial amount of communication, thus limiting their usability in large scale MAS. In a distributed system of reasonable complexity, the computation and communication costs of determining the optimal allocation far outweighs the improvement in the solution [Corkill and Lesser, 1986].

Most of the research in MAS has focused on finding approaches that impose a set of simple rules which individual agents have to follow locally without the need for global coordination. In Ephrati *et al.* [1995], the effects of introducing a filter-override mechanism to reduce the amount of required communication are studied by conducting a set of experiments in the Tileworld system. Shehory and Kraus [1998], and Learman and Shehory [2000] propose distributed algorithms of low complexity for the formation of coalitions, which is useful when a group of agents can be more efficient when working together or when no single agent by itself can satisfy a task. Shehory [2000] addresses the problem of locating agents without traditional approaches that require "middle" agents and thus impose infrastructure, protocol and communications overheads. The algorithm consists of an agent contacting only its neighbors who, in turn, can contact their own neighbors allowing information to propagate across the network.

Shehory *et al.* [1998] describe a Physics-based approach for distributed task allocation in the geographical domains that we address in this paper. Their solution is based on assumption that these domains are modeled well by a particle system governed by regular physics laws. While benefiting by being able to use many already developed formulations, this approach suffers from some limitations as well. In particular, only distance and not adjacency information is used, and it is not always beneficial for agents to repel each other. For example, they should not repel each other as they approach nearby tasks, or as they approach a task that requires multiple agents. In our system, we address this issue by having agents repel each other only in the absence of adjacent tasks. One limitation of our system is that in the presence of a dense cluster of task nodes, the agents will not be aware of task nodes that are only adjacent to other task nodes until those other task nodes are satisfied.

7. SUMMARY AND FUTURE WORK

We have presented an algorithm for efficient task allocation in distributed environments. Our approach uses computational geometry techniques to efficiently determine adjacency information for the agents. This adjacency information allows for agents to use a set of local rules to determine their course of action as they search for a nearby task to satisfy, therefore serving as an efficient filter for determining which neighboring nodes should be relevant in the decision making process.

The algorithms presented in this paper were implemented. We experimented with different system settings to observe how the system performed and to find optimal configurations. The algorithms yield improved task allocations compared to previous algorithms proposed for similar environments.

There are several interesting areas for future work:

- We only analyzed average results. One could analyze the distribution of these values to see whether some tasks wait much longer than others.
- One could experiment with heterogeneous agents (i.e., agents with different velocities).
- In this paper, we attempted to maximize the number of tasks fulfilled by agents. One could also consider other cost metrics, such as maximizing agent idle time. If the system is in equilibrium, each agent fulfills the same number of tasks per hour. In many cases all agents could considerably decrease their velocities while still maintaining equilibrium. In the extreme, every agent could remain idle for a period of time, in order to conserve resources. While this would increase the fulfillment time, it would not affect the average number of tasks fulfilled by the agents. So from the agents' point of view, this could be beneficial. Perhaps a learning algorithm could be used for the agents to determine when and for how long they should remain idle.
- Given this triangulation framework, agents can now easily propagate information by communicating to their few select neighbors. It would be interesting to try to develop a look-ahead algorithm for determining which task to pursue based on adjacency information.
- One could consider adopting some cooperation strategies aimed at increasing the common utility, and analyze how well cooperative agents will perform in the presence of selfish agents.

ACKNOWLEDGEMENTS

We are very grateful to Liz Bradley and Sarit Kraus for their suggestions and comments on earlier drafts. The research reported in this paper was supported in part by the National Science Foundation, grants IIS-9978343 and IRI-9618848 to Harvard University and a Microsoft Research graduate fellowship.

REFERENCES

[1] Barber, C. B., Huhdanpaa, H. The Qhull software library. URL: http://www.geom.umn.edu/software/qhull/

[2] Corkill, D. D., and Lesser, V. R. The use of meta-level Control for coordination in a distributed problem dolving network Proc. Int. Joint Conf. On AI, Karlsruhe, Germany, pp 748-756, 1986.

[3] Delaunay, B. Sur la sphère vide. Bull. Acad. Sci. USSR(VII), Classe Sci. Mat. Nat., 793-800, 1934.

[4] Ephrati, E., Pollack, M. E., Ur, S. Deriving multi-agent coordination through filtering strategies. Proceedings of 14th International Joint Conference on Artificial Intelligence, 1995.

[5] Gao, J., Guibas, L., Hershberger, J., Zhang, L., Zhu, A. Geometric Spanner for Routing in Mobile Networks. Mobihoc 2001.

[6] Jennings, N.R. Coordination Techniques for Distributed Artificial Intelligence. Foundations of Distributed Artificial Intelligence, 1996.

[7] Learman, K., Shehory, O. Coalition Formation for Large-Scale Electronic Markets Proceedings of the Fourth International Conference on Multiagent Systems, 2000.

[8] Malone, T. W. Modeling Coordination in Organizations and Markets. Management Science 33, pp 1317-1332, 1987.

[9] Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M. Coverage Problems in Wireless Ad-hoc Sensor Networks. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, 2001.

[10] Naor, M., Stockmeyer, L. What can be computed locally? 25th ACM Symposium on Theory of Computing, pp 184-193, 1993.

[11] Preparata, F., Shamos, M. I. Computational geometry: An introduction. Springer-Verlag New York Inc., 1985.

[12] Sandholm, T., Lesser, V. Issues in automated negotiation and electronic commerce: Extending the contract net framework. Proceedings of the First International Conference on Multi-agent Systems, 1995.

[13] Shehory, O., Kraus, S., Yadgar, O. Goal-satisfaction in large-scale agent-systems: a transportation example. Proceedings of the 5th International Workshop on Intelligent Agents V (ATAL-98), 1998.

[14] Shehory, O. A scalable agent location mechanism. In N.R. Jennings and Y. Lesperance, eds., Intelligent Agents VI - Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.

[15] Shehory, O., Kraus, S. Methods for task allocation via agent coalition formation. Artificial Intelligence, 1998.