Unstructured Agent Matchmaking: Experiments in Timing and Fuzzy Matching

Elth Ogston
Computer Engineering Laboratory, ITS
Delft University of Technology
elth@ce.et.tudelft.nl

Stamatis Vassiliadis
Computer Engineering Laboratory, ITS
Delft University of Technology
stamatis@ce.et.tudelft.nl

ABSTRACT

We investigate distributed matchmaking within an multi-agent system in which agents communicate in a peer-to-peer fashion with a limited set of neighbors. We compare the performance of a system with synchronized time to that of systems using several different models of continuous time. We find little difference between the two, indicating that the ordering of events does not play a part in computation. We also compare a system in which matches are made deterministically between discrete task categories to one in which task matches are made non-deterministically between continuous task categories. We consider several possible matching functions and show that their support is proportional to the spread of categories tolerable. This holds for matching probabilities as low as 0.01. We further show that the matching function's 'height' relates to the speed at which the system finds matches. For instance, we show that for a triangular matching function, doubling the probability of each service matching results in about a 1.6 times speedup.

Keywords

multi-agent systems, peer-to-peer computing, matchmaking

1. INTRODUCTION

In this paper we present results from two experiments designed to test how an abstract model of distributed, agent matchmaking, introduced in [5], might perform in a less precise real world setting. The model we investigate was first studied with the aim of determining if multi-agent systems could accomplish a matchmaking coordination task without being given a predefined system structure [5], and then later, without any form of global system structure [4]. These experiments showed some promising behavior. For example, agents communicating in a peer-to-peer fashion with a limited set of neighbors were able to find 90% or more of possible tasks matches in a short amount of time in systems with up to 100 categories of tasks. While promising however, these experiments leave some open questions. There is a danger that the ability to coordinate shown by the agents is due, in part, to the manner in which parallelism was simulated. The sequential order in which theoretically parallel agents actions are simulated may provide an unintentional means of regulation. In addition, the abstract nature of the tasks studied may have a simplifying effect that makes the problem solved by the agents much easier than any they might face in a more realistic situation. Thus, it could be the case that in a more true to life setting the system behaviors shown in the experiments presented in [4] and [5] will change dramatically.

In this paper we address these open questions. The experiments presented here are concerned with relaxing two assumptions made to enable analysis in the original model. The first was that all agents move at the same speed, thus allowing an order in which actions are performed to be defined. This method represents a form of global synchronizing time that will not exist in a truly distributed system. We test the model with several different methods of simulating continuous time and find that the order in which agents move in fact makes little overall difference. The second assumption that we put on trial is that the categories of tasks being matched are discrete and matchings themselves are deterministic. In actual applications this is unlikely to hold true. Agents may be looking for any of a number of different services to accomplish a task, some of which are better than others, and may probabilistically accept an offered service based on its suitability. We examine several forms of fuzzy matching and show that number of categories supported relates to how precise a match agents are willing to accept, even when given a low matching probability for less suitable services. Meanwhile, the probability of services matching is shown to affect the speed with which matches are found. Overall we find that the following may hold true:

- The system is robust with respect to timing; there is no dependency on the order in which agents move.
- The system is flexible with respect to how task categories are matched; agents do not need to know exactly what services they are searching for.

In the remainder of this paper we first discuss related work in sections 2 and the model we investigate in section 3. In section 4 we present simulation results and analysis. Section 5 concludes with some final remarks.

2. RELATED WORK

Within multi-agent systems research there are two commonly used methods of matchmaking; markets and facilita-

SAC 2002, Madrid, Spain

tors. In the first, economic models of commodity markets are used to 'trade' services among agents [9]. For instance, in the contract net protocol [7] agents broadcast a need for a service and service providing agents return bids for a contract. In general markets consist of a set of buying agents, a set of selling agents, and a central market. For efficiency's sake, agent markets often take the form of an auction in which bids and offers are sent to an auctioneer that broadcasts them to other agents in the system. These shouts are made in rounds, all agents placing shouts, listening to other's shouts, and then recalculating what they shall bid or offer in the next round. It has been shown that over time even agents with simple bidding strategies will converge to a market equilibrium price that is optimal for the system as a whole [6]. Facilitators, as generalized in the middle agents paradigm [2] [8], on the other hand rely on one central agent, or set of agents, to indicate where looked for services can be found [3]. Such systems create a central directory of service providers, service users, or both. This directory can then be queried by agents looking for services or clients.

In this paper we want, by contrast, to consider unstructured multi-agent systems. We study a model without any predefined structure like a central marketplace, broadcast mechanism or network of well know middle agents. Instead, our agents search for possible serve providers by making one-to-one queries among a small set of direct neighbors. Agent's search spaces are expanded by forming 'clusters' among agents that find they are able to work together. We have shown that this model will quickly organize into a single large cluster allowing 90% or more of matches to be found, provided that the number of task categories is limited [5]. We have further investigated limiting the cluster size in such systems and have found that while this reduces the percentage of matches found by around half, it still works well in the dynamic case where tasks are completed and agents go on to search for new task matches [4].

We are interested in investigating two possible unintentional sources of organization in this agent model; timing and task category structure. Global clocks are often used for synchronization. Knowing the order in which events will take place can allow a system designer to make assumptions in calculations. In auctions the fact that bids take place in rounds allows agents to update their shouts based on a know current best bid or offer. This speeds-up the time it takes the market to converge to equilibrium. In a system that uses a middle agent directory the middle agents can control the order in which requests are handled. They can thus distribute requests to service providers based on their capacity to handle them. When attempting to create a fully distributed agent system we must be sure that none of the system behavior observed is due to such a hidden dependency on timing.

The method by which task categories are matched can also provide an unseen form of structure. Many agent markets deal with commodity items, meaning that all offers and bids for particular services are interchangeable; different services are traded in different markets. Middle agent directories are more flexible; they provide a central ontology that allows all services in the system to be described in the same manner, simplifying matching decisions. However, among a distributed group of autonomous individuals this strict ontology is unlikely to exist. Individuals may not know an exact definition of a desired service and can make choices based on a subjective judgment of suitability. This means that one-to-one negotiations are likely to be non-deterministic as individuals can consider more decision factors than can be communicated to a middle agent. For this reason we look at fuzzy matching among task categories. As an example, imagine that I would like to purchase a book on java programming. If I knew that I wanted the book with ISBN number 1-56592-262-X I could place a bid in a Kasbah [1] auction or order it from any bookstore in my local yellow pages. On the other hand if I simply want a book on Java and browse through several of my local bookstores I can consider many other factors in my decision of which book to purchase; my general impression of the book's layout, or the niceness of the staff, for instance. This manner of finding a book can easily lead me to purchasing a different book from a different store over several trials of the experiment.

3. OUR MODEL

In this section we define the model we are working with and describe how it differs from the simpler version used in previous work. We further give a brief overview of previous results about the model's basic behaviors. The model presented is focused on providing an abstract representation of agents attempting to find partners for tasks using peer-to-peer communications. Our aim in its design is to simulate a system that will help us determine under what conditions unorganized agents can find partners, and to minimize any predefined means of cooperating. We consider agents that each have a number of tasks that they would like to find partners for. Each tasks has a category which determines what partner tasks are suitable; each task category has one or more matching categories. Agent's tasks are initially linked at random with a neighbor task. Agents then search for matches for their tasks. They do this by moving in turns in which they reassign their links, shuffling which of their unmatched tasks are paired with which neighbor task. When a match is found it forms a connection between the agents of the tasks involved. Connected agents act as a single entity and move by shuffling all of their unmatched tasks and neighbors together. Creating clusters in this way expands the search spaces of the individual agents, allowing them to find partners outside of their initial neighborhood.

Our system consists of a set of agents $A = \{a_1, \ldots, a_n\}$, and a continuum of task categories $C = [0, m)$. These tasks categories represent the types of job for which an agent may seek a partner. We suppose that there is a distance $d : C \times C \rightarrow [0, \infty)$ on C. The distance between two categories measures how similar they are. In this paper we use a simple cyclic distance: $d(c, c') = \min\{|c - c'|, m - |c - c'|\}$. This cyclic distance is used to avoid edge effects, and gives a maximum distance of m/2. Furthermore, we consider a matching function on the possible distances: $f : [0, m/2] \rightarrow [0, 1]$, where $f(d)$ is the probability that two categories at distance d form a matching pair. This function determines if an agent looking for a service of category c will accept a candidate one of category c'. We consider functions f that are non-increasing and have a maximum at 0, representing the fact that agents are more likely to accept offered services the more similar they are to the one desired.

In our model, each agent a in A has a set of k tasks $T_a = \{t_1, \ldots, t_k\}$, each task belonging to a category in C. Note that T_a can contain more than one task from a category. The goal of the matchmaking problem is for the agents to

create links between their tasks and those of other agents, maximizing the number of links between matching client-server pairs as defined above. We shall represent this by defining a graph $G = (V, E)$ where the nodes are all of the agents' tasks and edges are placed between any tasks that have a non zero probability of matching. More formally:

$V = \{(a,t) : a \in A \text{ and } t \in T_a\}$ and

$E = \{(u,v) \in V \times V : u = (a,t), v = (b,t') \text{ such that if } c$ is the category of task t and c' is the category of task t', then $f(d(c,c')) > 0$.

Thus G is a graph representing all the possible matches in the system. The aim of the matchmaking problem is to approximate a maximum sized matching M in G, i.e. a set $M \subset E$ such that no two edges in M are adjacent. This represents a system where each task is paired to one other task and the number of matching client-server pairs is approximately maximized.

We initiate our simulations of this model by creating agents with tasks chosen uniformly at random. We then take a random matching of size $[|V|/2]$ in the complete graph on V so that each task in each agent is linked to one other task in another randomly chosen agent. This represents initial neighborhoods formed by some means outside of the system, for instance based on location. The initial links are tested to see if they form connections, i.e. that they represent a matching pair of tasks. Agents then search for further links that are members of E above, i.e. they look for links that are between two possibly matching tasks. When a new link is formed (either initially or in the later searching process) the matching function f is used as the probability of that link becoming a connection.

Agents search for connections by permuting which of their unmatched tasks are paired to which of their unmatched neighbors. This is done in turns; during each turn each agent randomly reassigns all of its unmatched tasks to its unmatched neighbor tasks. For each of these new links f is used to determine if a new connection is created. A connected link represent two agents agreeing to cooperate. We assume that agents that cooperate on one task are likely to be able to work together more closely, for instance they might represent devices from the same manufacturer. Thus, we group such connected agents into a cluster. Clusters then act like compound agents; the unmatched tasks and neighbors of the agents in the cluster are all shuffled together on a turn. Simulations halt after no new connections are formed within a set number of turns.

In a previous study [4] [5] we investigated a discrete deterministic version of the system described above, one in which there are a finite number of categories represented by integers and the matching function is 1 at distance 0 and 0 otherwise. Three measures were used to characterize the general behavior of this system: the number of categories supported, the percentage of connected links at the end of a trial, and the number of turns until a trial stops changing. It was found that for low numbers of categories trials will always connect into a single cluster, with high numbers of categories they will never connect, and between there is a steep drop where some percentage of trials connect. The percentage of connecting trials as a function of the number of categories was graphed to determine the point where this drop begins and the curve's steepness. It was found that for agents with 3 tasks each the percentage of connecting

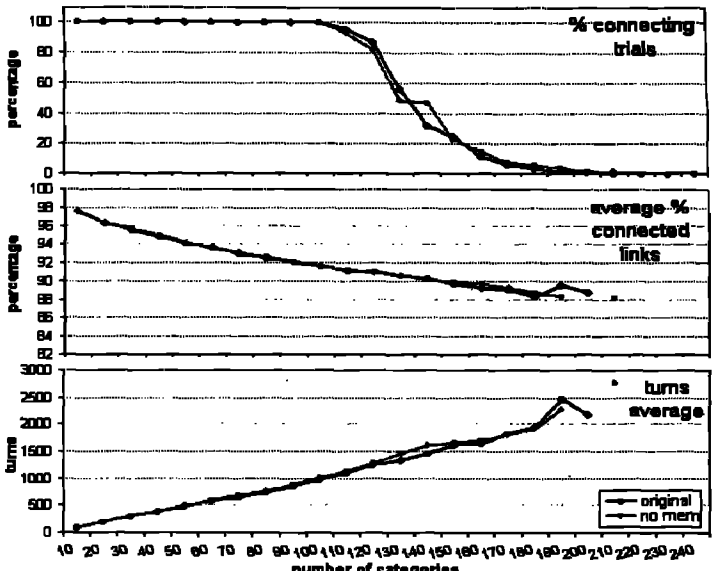

Figure 1: original synchronized system vs. no memory unsynchronized system

trials remains 100% until around 100 categories, and then drops to 0% at about 200 categories. Further, the number of categories supported increases to 400 categories when each agent is given 4 tasks and 800 categories with 5 tasks per agent. For the trials that formed a single cluster the percentage of tasks within the system that are matched to partners was also investigated. This was found to be around 97% for systems with 10 categories, dropping to 90% at 200 categories. Finally the number of turns that connecting trials ran for before they stopped changing was graphed. This ranged from an average of 298 at 10 categories to 2369 at 200 categories. These previous results are from simulations run with 2000 agents each with 3 tasks, and will be used for comparison with the experiments presented here. We have also studied more complicated systems with limited cluster sizes and tasks that end and get reassigned. These however are more difficult to analyze and as their basic behavior relates closely to that of the static, large cluster system we choose to look only at the simpler system in the following experiments.

4. RESULTS

In the following sections we present results from two experiments; one comparing performance of the original system with different approximations of continuous time and a second looking at forms of matching functions for systems with continuous task categories. The experiments presented below all use 2000 agents each with 3 tasks, a system small enough to be quickly simulated yet large enough that a lack of agents doesn't affect system behavior. Trials are run until no change occurs within 200 turns, and the last turn on which a change was made is recorded as the end of the trial. For graphs of the percentage of trials that connect we ran 100 trials at each category point.

4.1 Continuous Time

We first want to ensure that the behavior observed in the original experiments is not due to any structure in the order in which agents move. The original experiments were run in turns, each agent or cluster moving once each turn, in the

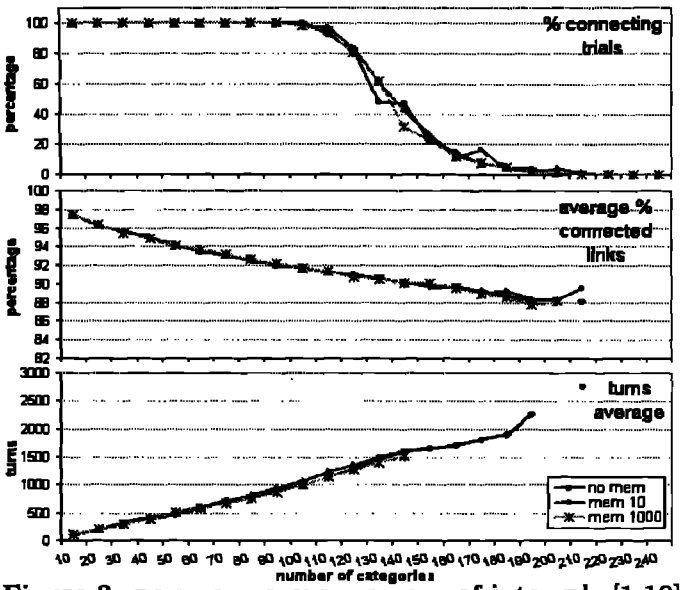

Figure 2: no memory v.s memory of intervals [1,10] and [1,1000]

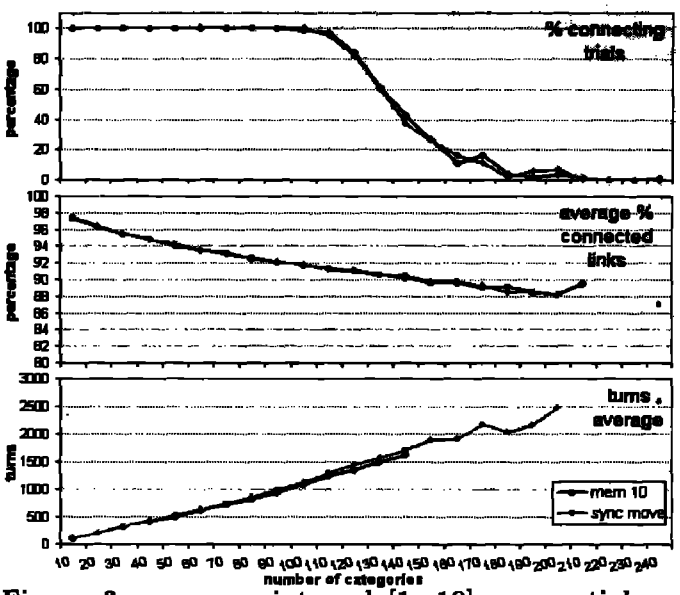

Figure 3: memory interval [1, 10]; sequential vs. synchronous cluster movement

order in which they are stored in an internal array. There are several ways to simulate continuous time on a discrete computer. In the following sections we study a system with no memory, and several where clusters wait some period of time between moves. As we wish to compare to the original system we consider the discrete deterministic matching case: categories have integer values between 0 and m and the matching function is 1 at distance 0 and 0 otherwise. We use a distance function such that each category matches to exactly one other category.

4.1.1 *No Memory*

One way of simulating an unsynchronized system is to move clusters in a randomly chosen order. Figure 1 compares the original system (described above) to one in which moves are made one cluster at a time, each step choosing which to move uniformly from all existing clusters. For comparison's sake we mark turns as ending after N moves, where N is the number of clusters at the start of the turn. We see that there is little difference between the two. The unsynchronized system stops reliably forming large clusters with slightly fewer categories, however the percentage of connections in its end clusters and the length of trials are almost identical.

4.1.2 *Memory*

A more realistic version of an unsynchronized system considers the fact that a cluster that has recently moved is less likely to move next then others that have been still, and thus places some time between the moves of a cluster. We simulate this by having each cluster wait some number of time steps, uniformly chosen from an interval $[t_{min}, t_{max}]$, between each move. We first consider a situation in which clusters that wish to move simultaneously in a given time step are moved one at a time in a random order. Figure 2 compares the no memory case in Figure 1 to two versions of such a system, "mem 10" with wait times of between 1 and 10 steps, and "mem 1000" with waits of between 1 and 1000 steps. For comparisons sake we measure turns after 10 or 1000 time steps respectively, the expected amount of time for each cluster to move once. Since we have 2000 agents, and on average even fewer clusters, we expect the "mem 1000" case to approximate the no memory case, as we see occurs in the graphs. The "mem 10" case however also shows no appreciable difference indicating that this change in the order of moves makes no difference to the overall system.

4.1.3 *Moving in Unison*

Our previous cases clusters move sequentially. More realistically we want to consider clusters that move in parallel. If moved in sequence two neighboring clusters will get to test two possible matching task pairs, first one cluster moves and tests for a match, then the second moves, creating a new pair, and tests again. Two clusters that move at the same time will skip the intermediate task pair, testing only one new pair. Figure 3 compares the "mem 10" experiment above with an experiment run with the same memory parameters but moves made in this synchronous manner. As expected the system with synchronous moves takes slightly longer (2.4% to 12.3%) due to this single testing, but otherwise we see little difference between the two methods.

4.2 Continuous Categories

In our second experiment we study the difference between systems with discrete categories and deterministic matches, and systems with categories chosen from an interval with various matching functions. Our categories are now chosen from an interval $[0, m)$, and we look at matching functions on the distance between categories, $f(d)$ as defined in Section 3. In the following sections we look at several different matching functions, first to compare against the discrete case, and then to determine the effect of changing their width (i.e. the support of $f; S_f = \{d : f(d) > 0\}$), average height or fuzziness.

4.2.1 *Original vs. Rectangular*

We first compare the original discrete case to its closest match in a continuous system. Figure 4 compares the original data with a matching function that is 1 between 0 and 1/2 and 0 everywhere else, labelled 'rectangular'. As in the

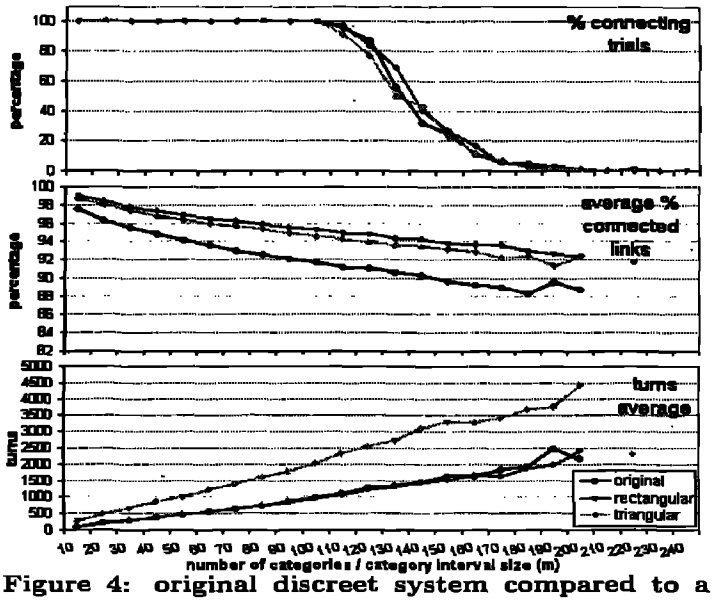

Figure 4: original discreet system compared to a continuous systems with 'rectangular' and 'triangular' matching functions

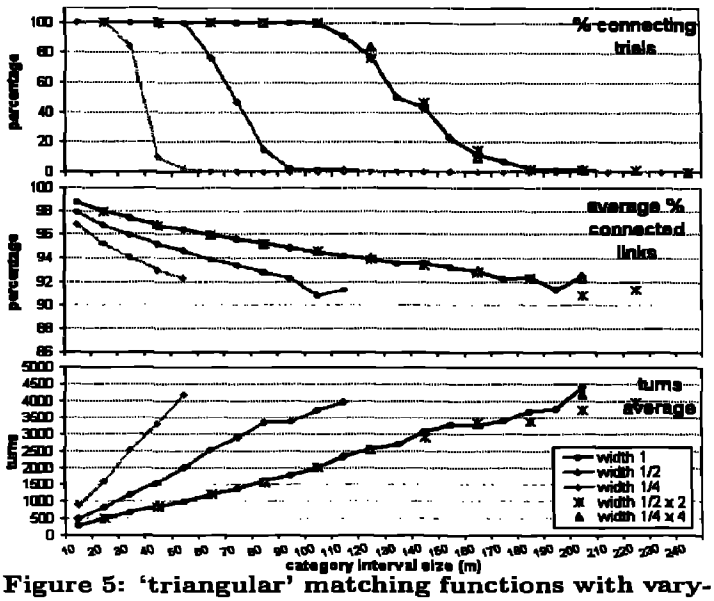

Figure 5: 'triangular' matching functions with varying widths

original system, this 'rectangular' function represents a case where a category will always match to any category within a distance of 1/2 to either side. We see that the main difference is that the 'rectangular' case improves on the percentage of connected links at the end of a trial by 1.4% to 5.3%. This is due to the fact that if a task A has two matches, B and C, B and C are unlikely to be identical and thus while some of their matching space overlaps there are also further tasks that will only match B or only match C. Thus B connecting to A removes less of C's potential matches then it does in the discrete case.

4.2.2 *Triangles*

In a more realistic system it is likely that an agent is willing to accept a number of different related services as modelled in the 'rectangular' case above, but also that the agent will prefer some of these services to others. In the following section we model this behavior using triangular matching functions. These 'triangular' functions provide a simple representation of an agent's preference for a particular category, with a diminishing ability to accept related categories depending on how similar they are to the desired category. They place the highest probability of matching at distance 0, and then linearly lower the matching probability to 0 at some distance d_{max}. For comparison to the 'rectangular' case, Figure 4 includes data for such a 'triangular' function with $f(0) = 1$ and $d_{max} = 1/2$, labelled 'triangular'. We see that the 'triangular' case in general performs less well than the 'rectangular'. It supports the same number of categories, but with a more gradual drop off and end clusters have 0.1% to 1.5% less connected links. More importantly however, runs are 1.8 to 2.9 times longer, due to the fact that pairs that may possibly connect might meet a number of times before doing so. We further explore two sets of these 'triangular' functions, first holding $f(0)$ at 1 and varying d_{max} to determine the effect of the 'width' or support of the matching function. We then fix d_{max} at 1/2 and vary the height of the triangle to determine the effect of the probability of two tasks matching.

Figure 5 shows the first of these experiments. All data sets show 'triangular' matching functions with $f(0) = 1$; "width 1", "width 1/2", "width 1/4" have $d_{max} = 1/2$, 1/4, and 1/8 respectively. (Intuitively 'width' 1 indicates that each category can possibly match to an interval of categories of length 1) The points labelled "width $1/2 \times 2$" show the data for "width 1/2" with its category axis doubled, similarly "width $1/4 \times 4$" shows the data for "width 1/4" with its category axis quadrupled. As these points indicate multiplying the width of the matching function by a constant simply has the effect of multiplying the supported category range by the same constant. This makes sense, categories are chosen from a continuous interval so scaling the matching function's support is equivalent to scaling the interval. We next look at the effect of varying the height of a 'triangular' matching function. In Figure 6 all data sets have a 'triangular' function with $d_{max} = 1/2$. "height 1", "height 1/2" and "height 1/4" have $f(0) = 1$, 1/2 and 1/4 respectively. Reducing the probability of matches being made has a small effect in reducing the system's ability to connect, but most importantly changes the speed at which the system finds matches. Compared to "height 1", "height 1/2" takes 1.5 to 1.7 times longer and "height 1/4" takes 2.4 to 2.8 times longer. Thus doubling the height gives an approximately 1.6 times speedup to the system.

4.2.3 *Fuzzy Tails*

In our last experiment we look at the effect of 'fuzziness' in our system. As the 'triangle' graphs indicate, even a very low probability of matching at some distance can contribute to the overall system behavior. Though the 'triangular' cases took longer to run than the 'rectangular' one, they still supported about the same range of task categories in spite of their low edge probabilities. To look further at this we want to consider a case where agents have a preferred match for a task, but could possibly take matches that are very different from their preferred partner. We use the unit normal distribution function to model this behavior as it has a high center and very low probability tails. We scaled the width of the unit normal by 2/11 to give approximately the

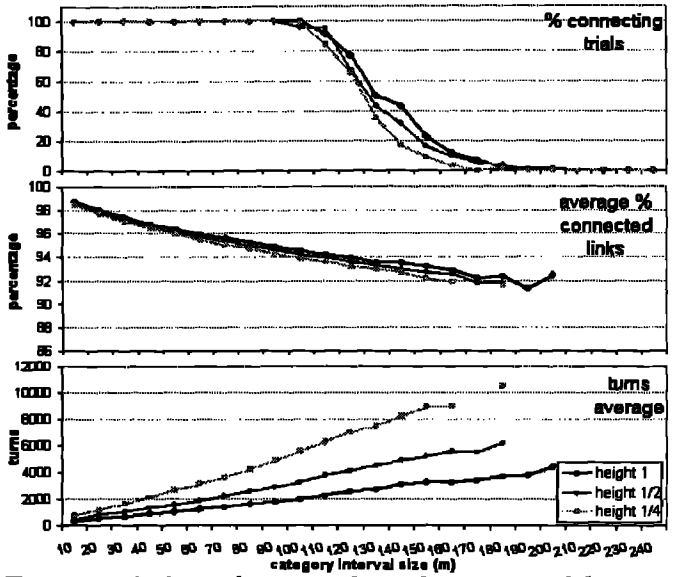

Figure 6: 'triangular' matching functions with varying heights

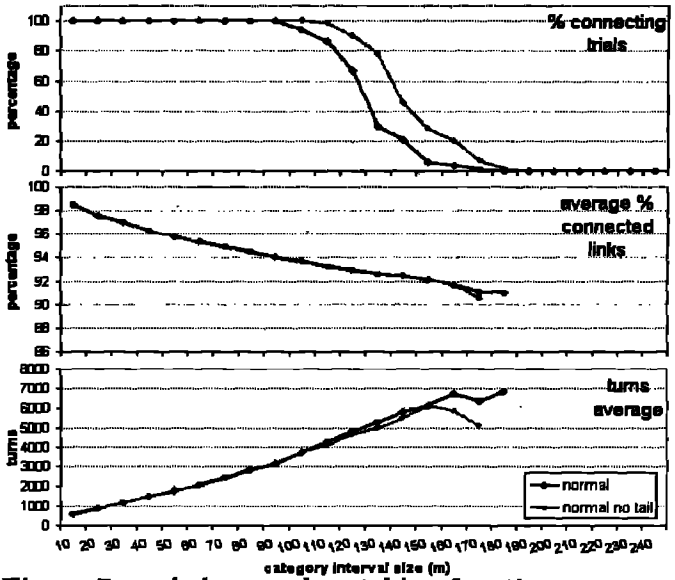

Figure 7: scaled normal matching function vs. same cut off at 1/2

same range of supported categories as our previous experiments. This is the "normal" data in figure 7. The "normal no tail" data shows this same function but with probability at distances [1/2, infinity] set to 0. At distance 1/2 the probability of two tasks matching is already very low so the cut off tail accounts for only 1% of the total area under the original curve. However cutting off this tail still affects the number of categories supported by around 12%, though it makes little difference to the length of runs or the quality of solutions.

5. CONCLUSION

In this paper we considered matchmaking in an unstructured multi-agent system. Using simulations we have shown that the following holds true:

- We tested several methods of simulating continuous time and found no dependency on the order in which clusters move. This held true for systems with and without memory, and for systems that assumed that clusters move in parallel.
- We found that the system is flexible with respect to how task categories are matched. Agents can be unsure of what matches they are willing to accept and can accept a range of matches, even with probabilities as low as 0.01. The range of categories accepted in this manner is proportional to categories supported by the system as a whole, and the average probability of categories in this range being accepted relates to speed of the system. A doubling in height of a triangular function results in roughly a 1.6 times speedup.

These experiments, while still abstract, indicate that uncontrolled systems of interacting agents might be able to maintain stable behaviors, in spite of the complications thrown at them by real world applications. There are however some limitations that may effect the suitability of the system we've described for a given application. The range of categories supported by our system is limited while the category range needed in an actual problem, especially in problems that consider multi-dimensional categories, could be very large. Further, the speed of a 'turn' depends upon the communication costs faced by the agents. These issues constitute further research directions.

6. REFERENCES

[1] Chavez, A., Maes, P.: Kasbah: An Agent Marketplace for Buying and Selling Goods. Proceedings 1st Int. Conference on the Practical Appication of Intelligent Agents and Multi-Agent Technology. (1996) 75-90

[2] Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. Proceedings of the 15th Int. Joint Conference on Artificial Intelligence. (1997) 578–583

[3] Kuokka, D., Harada, L.: Matchmaking for Information Agents. Proceedings of the 14th International Joint Conference on Artificial Intelligence. (1995) 672-678

[4] Ogston, E.,Vassiliadis, S.: Local Distributed Agent Matchmaking. Proc. 9th International Conference on Cooperative Information Systems . (2001) 67-79

[5] Ogston, E.,Vassiliadis, S.: Matchmaking Among Minimal Agents Without a Facilitator. Proc. 5th Int. Conference on Autonomous Agents.(2001) 608-615

[6] Preist, C., Van Tol, M.: Adaptive Agents in a Persistent Shout Double Auction. Proc. 1st International Conference on the Internet, Computing and Economics. (1998) 11-17

[7] Smith, R.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Trans. On Computers, 29(12). (1980) 1104-1113

[8] Sycara, K., Lu, J., Klusch, M., Widoff, S.: Matchmaking among Heterogeneous Agents on the Internet. Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace. (1999)

[9] Vulkan, N., Jennings, N.: Efficient Mechanisms for the Supply of Services in Multi-Agent Environments. Int. Journal of Decision Support Systems, 28(1-2) (2000) 5-19