Local Distributed Agent Matchmaking

Elth Ogston and Stamatis Vassiliadis

Computer Engineering Laboratory, ITS, TU Delft, Delft, The Netherlands
{elth, stamatis}@ce.et.tudelft.nl

Abstract. This paper explores through simulation an abstract model of distributed matchmaking within multi-agent systems. We show that under certain conditions agents can find matches for cooperative tasks without the help of a predefined organization or central facilitator. We achieve this by having each agent search for partners among a small changing set of neighbors. We work with a system where agents look for any one of a number of identical task matches, and where the number of categories of tasks can be as large as 100. Agents dynamically form clusters 10 to 100 agents in size within which agents cooperate by exchanging addresses of non-matching neighbors. We find that control of these clusters cannot be easily distributed, but that distribution in the system as a whole can be maintained by limiting cluster size. We further show that in a dynamic system where tasks end and clusters change matchmaking can continue indefinitely organizing into new sets of clusters, as long as some agents are willing to be flexible and abandon tasks they cannot find matches for. We show that in this case unmatched tasks can have a probability as low as .00005 of being changed per turn.

1 Introduction

In this work we are interested in the global behavior of systems of interacting entities, where interactions take place on a solely local level, and entities make non-deterministic decisions. We consider this setting a good abstract model for large multi-agent systems. [3] Our purpose in studying such systems is to characterize on a conceptual level the behaviors possible in multi-agent systems. We would like our model to include the following properties:

- no agent or other system entity has a global view of the entire system
- agents each know about and communicate with only a small subset of the rest of the system
- agents are built by many different designers
- the system is open and dynamic; agents come, go and change purpose often
- agents are initially placed into the system without considering a global view of the system

We limit the agents in our model to performing exclusively local interactions. By this we mean that agents can only interact within a local neighborhood; that is a small subset of other agents whose addresses are known to that agent. In

addition no agent may require any form of knowledge about the entire system, thus we cannot have any central controlling entities. The size of an agent's neighborhood is based upon the resources of the agent, however what we consider important is that as the number of agents in the system grows the size of the neighborhood of any agent in the system remains constant. We create this restriction for a number of reasons. First, it creates a more scalable agent system. Broadcast based algorithms that are required to maintain system wide information are expensive in terms of communication for very large systems. Single node broadcast takes O(r) time, where r is the diameter of the network. In addition to the cost, there are a number of reasons to avoid global information that relate to agent autonomy. Maintaining global information requires cooperation. Thus it requires agents to act on a scope that considers the system as a whole, whereas agents that consider only their individual interests are simpler and much easier to create. Additionally global information is open to sabotage as a malicious agent could corrupt the data that other agents rely on.

A second characteristic of our model is that choices are made nondeterministically. By "nondeterministic" we mean to specify that from the system level view such agent algorithms are forms of randomized algorithms. Agents are modelled as making random choices, though this can include some measure of the agent's environment. Again, this restriction is used to denote several phenomena. First it represents the autonomy of the individual agents; if an agent is truly autonomous we cannot predict how it will behave when observing it from a system level. Second, it represents agent choices that may be made based on incomplete or incorrect data. An agent may have a fixed decision behavior, but may make the wrong choices because it is not possible to for it to know with certainty the data that those choices are based on. Finally, it represents the possible stupidity of agents. Agents may have correct data, yet be limited in their reasoning powers, or even poorly programmed and thus make unexpected choices.

Overall we would like to know what kinds of algorithms a system limited as described above is capable of performing. In this paper we consider a specific problem from this point of view, a form of the matchmaking problem. Matchmaking asks how agents requiring outside services find other agents who are capable of providing those services. We choose the matchmaking procedure for several reasons: it is an essential component of many agent systems, it is well studied, and the majority of algorithms for matchmaking use a central or broadcast component. In a previous paper we defined a local random search procedure for matchmaking and showed that it performed well under a wide range of parameters. [6] However our model had a major drawback in terms of locality; agents were grouped into clusters and their search space was limited to the neighbors of their clusters. Clusters were controlled centrally, and as clusters could grow as large as all the agents in the system, a cluster controller could become a global central controller. In this paper we show that cluster operations cannot be easily distributed, however we find that the size of clusters can be limited. We also extend our model to a dynamic system where tasks complete and agents continue

to search for partners for further tasks. We show that such a system deteriorates over time if agents make completely random choices for their new tasks, but can continue indefinitely if agents are flexible and are willing to abandon tasks or consider the partners available when choosing new tasks.

In the remainder of this paper we first discuss related work in sections 2 and our model in section 3. In section 4 we present a summary of our previous results with this model. Section 5 presents new results on decentralizing clusters, limiting cluster size and task completion. Section 6 concludes with some final remarks.

2 Related Work

Matchmaking within multi-agent systems is generally accomplished through a market or by providing middle agents. For one-to-many or many-to-many negotiations market mechanisms are used, as in Contract Net [9]. Here an agent requiring a service broadcasts a request, and providers of that service return bids. Repeated bids and offers can then be used to negotiate fair prices. Markets however can require a large amount of communications as bids and offers are often broadcast to all participants. Moreover, these broadcasts are commonly made though a central market controller that must keep track of all market participants [11]. The second common form of matchmaking within multi-agent systems is through providing a broker or middle agent or agents [2]. There are many different forms of such agents, but they all are based on a set of agents that act as a directory for services available in the system. These directory agents are well know to the other agents, and are queried to find matches. [4] [10]. However this means that middle agents must provide a directory of the entire system. For a large and dynamic system such a directory can be expensive to maintain, and if there are many requests for matches the middle agents can become a bottleneck. On the other hand they provide an efficient solution in terms of numbers of messages. Further, they can guarantee that an agent finds a match if it exists or allow an agent to find the best match system wide. In addition to work on markets and middle agents, there is also a good deal of work on more abstract models of matchmaking. This work is focused on defining the exact cost of creating and maintaining directory servers, and on the levels of distribution possible [1] [5].

In this work we are more curious about how matchmaking can be done on a purely disorganized local level. We explore to what degree matchmaking is still possible among agents that only have knowledge of a few direct neighbors. We find this question interesting as it gives an indication of how well multi-agent systems can organize by themselves, versus how much structure they must have pre-defined for them. It also gives an indication of how well very large unregulated systems might be able to perform from the point of view of an individual agent. One model for such a system is acquaintance networks as described in [7]. This model replaces central directories with broadcast requests made by each agent. We attempt to improve on this by removing the broadcast element, which

requires agents to pass on messages for each other, replacing it with a changing neighbors and clustering routine. A related area is the formation of coalitions [8], though again this is mostly focused on forming optimal coalitions and thus also uses system wide broadcast.

3 Model Definition

The model we use in this work in focused on providing an abstract representation of agents attempting to find partners for tasks. Our purpose is to determine under what conditions unorganized agents can find partners, and to minimize any predefined means of cooperating. We would like to determine if such mechanisms are possible given reasonable processing time and memory limitations on individual agents.

Our system consists of a set of agents $A = \{a_1, \ldots, a_n\}$, and a set of task categories $C = \{c_1, \ldots, c_m\}$. Each member of the set of tasks categories, c_i, represents a type of job for which an agent may seek a partner. We further define a pairing among these categories: $f : C \times C \rightarrow [0, 1]$ with

$$f(c_i, c_j) = \begin{cases} 1, \text{ if } (c_i, c_j) \text{ is a matching pair} \\ 0, \text{ otherwise.} \end{cases}$$

We consider the case where each category has only one match and that matches are symmetric. More precisely:

$$\sum_{j=1}^{m} f(c_i, c_j) = 1 \text{ for all } i,$$
$$\sum_{i=1}^{m} f(c_i, c_j) = 1 \text{ for all } j,$$

$$\text{and if } f(c_i, c_j) = 1, \text{ then } f(c_j, c_i) = 1.$$

If $f(c_i, c_j) = 1$ and $i = j$ we have a job that requires the cooperation of two like tasks. More interestingly, we concentrate on the case where $i \neq j$, representing a client-server pair. We show later that these two cases can produce some significantly different behaviors.

In our model, each agent a in A has a set of tasks $T_a = \{t_1, \ldots, t_k\}$, each task belonging to a category in C. 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 matching tasks. More formally:

$$V = \{(a,t) : a \in A \text{ and } t \in T_a\} \quad \text{and}$$
$$E = \{(u,v) \in V \times V : u = (a,t), v = (b,t') \text{ such that if } c \text{ is the category of task } t \text{ and } c' \text{ is the category of task } t', \text{ then } f(c,c') = 1\}.$$

Thus G is a graph representing all the possible matches in the system. The matchmaking problem is to find a matching in G, i.e. a set $S \subset E$ such that no two edges in S are adjacent, of maximal size. This represents a system where each task is paired to one other task and the number of matching client-server pairs is maximized. In this work we consider an approximation algorithm for this matchmaking task and instead of searching for a maximum size matching we look for a sufficiently large one.

In our model we initially create random links between tasks. This creates a graph that consists of a random subset of the edges in the complete graph on V above, such that no two edges are adjacent and the degree of each node is 1. In other words each task in each agent is linked to one other task in another randomly chosen agent, giving the agents one neighbor for each task. This represents starting connections formed by some means outside of the system, for instance based on location. Agents then start searching for links that are members of E above, i.e. they look for links that are between two matching tasks. Such connected links represent two agents being able to cooperate. 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 reassigns all of its unmatched tasks and links. When a connection is formed, the agents involved are considered able to cooperate. We assume that agents that can 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.

4 Summary of previous results

In a previous paper we reported on the basic behavior of the model described above [6]. Behavior was determined through simulation as the system is too complicated for an exact analysis. Shuffling was done in fixed turns, each agent or cluster shuffling once per turn. During a shuffle links within an agent or cluster were rematched at random with equal probability. In general we found that such a system either quickly formed a single large cluster where almost all links were matches, or remained almost completely unconnected. This behavior varied with the number of categories. An example is shown in Figure 1; for

10 categories almost all matches are found quickly, with 80 categories matches are found quickly, but after an initial slow phase, and with 300 categories most matches are never found. As we increased the number of task categories the probability of this single cluster forming during a trial remained 100 percent until a point at about 90 categories where it drops off suddenly to near 0 percent. We found that this drop off point increases to around 400 categories when each agent is given 4 tasks and 800 categories with 5 tasks per agent. Finally we showed that the system scales well in the number of turns as the number of agents in the system is increased. With 120 categories 2000 agents find clusters in 900 turns while 32000 agents require 1500 turns. For comparison Figures 3, 4 and 5 show data from these original experiments, labelled "original".

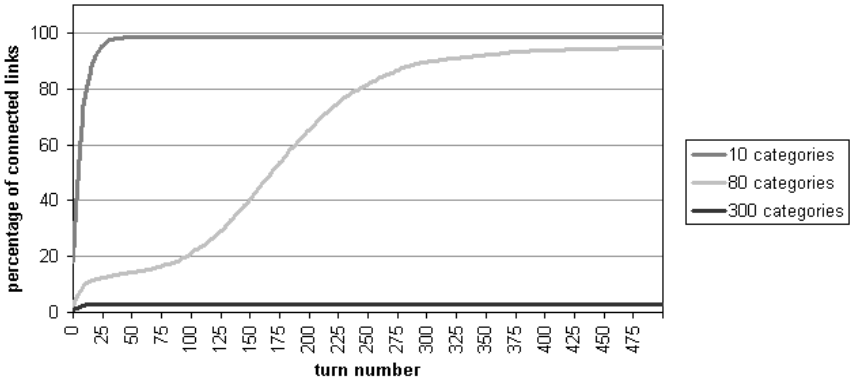

Fig. 1. Sample trials - 2000 agents; 10, 80 and 300 categories.

5 Results

Our previous work considered a system where each cluster was controlled centrally. Thus a single entity kept track of the unmatched tasks of a cluster and was responsible for shuffling and reassigning links. As clusters grew in size the number of unmatched tasks in the cluster also increased. Its maximum value approaches 1/3 to 1/2 of all the tasks in the system. Thus the memory need by the central element of a cluster and the time to do a shuffle could become proportional to the number of agents. In the following sections we explore solutions to this problem. We first show in section 5.1 that decentralizing the clusters is not feasible. However, in section 5.2 we find that we can limit the maximum size of clusters and thus keep the system as a whole decentralized. In section 5.3 we study a dynamic system where tasks end and show it always eventually decays. We remedy this in section 5.4 by adding a probability of agents abandoning unmatched tasks. Finally in section 5.5 we briefly describe an unsynchronized

version of these experiments. The experiments shown below all are for 2000 agents, each with 3 tasks. All trials are run until the system remains unchanged for 200 turns.

5.1 Decentralizing clusters

We wish to create a system that remains local, keeping the number of agents each entity in the system knows about and the time any operation takes independent of the number of agents. One way of doing this would be to distribute the control of each cluster. We can change the shuffle operation so that instead of randomly mixing tasks within a cluster, unmatched tasks are given a cyclic ordering and neighbors are simply passed from $task_i$ to $task_{i+1}$ each turn. Experiments using this form of shuffling are labelled "rotate" in Figures 3, 4 and 5 which show that the system still performs well with this form of shuffling. Figure 3 shows, as described above, that our rotating shuffle supports about 20 categories fewer than the original system. Figure 4 shows the percentages of links connected, on average, at the end of the runs in Figure 3. Values for trials that formed a large cluster are shown separately as the "high" data points, and those for trials that did not connect are labelled "low". Here the rotating shuffle performs as well as the original system. Finally Figure 5 shows the number of turns required on average to complete trials that connected. Here the rotate shuffle performs between 14% and 30% better than the original system.

While it is easy to decentralize the rotate operation, we run into problems when forming new matches. When combining two cluster's task sets, A and B, to form a new connection between tasks a_i and b_j a new order can be created by two different methods, as shown in Figure 2. In "Method 1" a_i and b_j are removed leaving the order a_{i-1}, b_{j+1},...,b_{j-1}, a_{i+1}. Alternatively in "Method 2" the new order is $a_{i-1}, a_{i+1}, \ldots, b_{i-1}, b_{i+1}$. As Figure 2 shows, which of these methods creates a single cycle depends on whether the connection being formed is between two tasks in the same cluster or tasks in different clusters. Without a cluster center this cannot be easily determined. A broadcast message could be sent through the clusters either asking if the connecting agents are in the same cluster or giving a new cluster ID to agents in one of the clusters. Alternatively the direction of one of the orderings can be reversed, thus creating an identical connection method for each case. However, this takes time proportional to the number of the free tasks in that cluster. Additionally we have further problems if we wish to break a connected link and must know if the cluster it is in remains connected.

5.2 Limiting cluster size

Another approach to maintaining locality in our model is to keep clusters centralized, but to limit the size of the clusters. One way to do this would be to connect every matching task pair when a new cluster forms, thus limiting the number of unconnected tasks. This works if matches are made so that category c_i matches itself. However, if matches are between client-server pairs it still leads

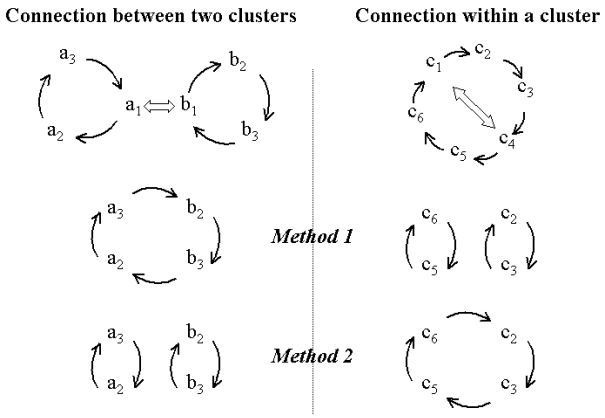

Fig. 2. two methods of connecting distributed clusters.

to large clusters as clusters build up multiple copies of either the client or server in each pair. We found that clusters reached a maximum of between 100 and 300 unconnected tasks when running trials with this method of connection.

In our original experiments we considered only cluster formation, not systems where tasks end thus break up clusters again. Another way of limiting the maximum cluster size would be to adjust the rate at which tasks end so that clusters fall apart before becoming too large. However, the evenness of the rate at which connections are formed makes this adjustment difficult. A high disconnect rates stops most clusters from forming whereas a slightly lower one still allows for large clusters.

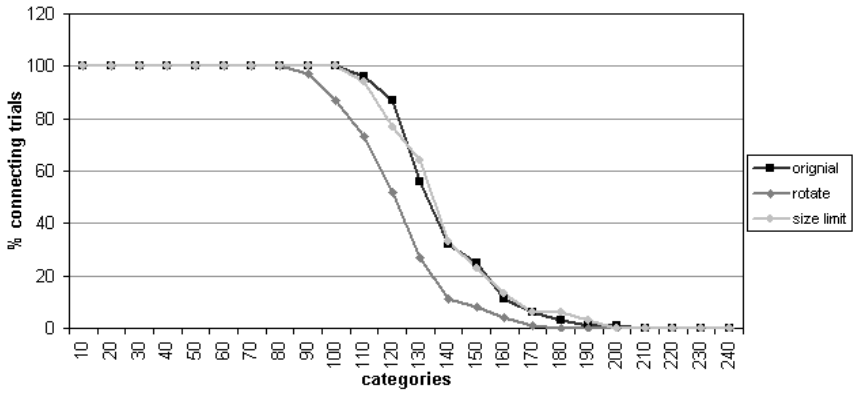

Fig. 3. percentage of connecting trials: 100 trials per category

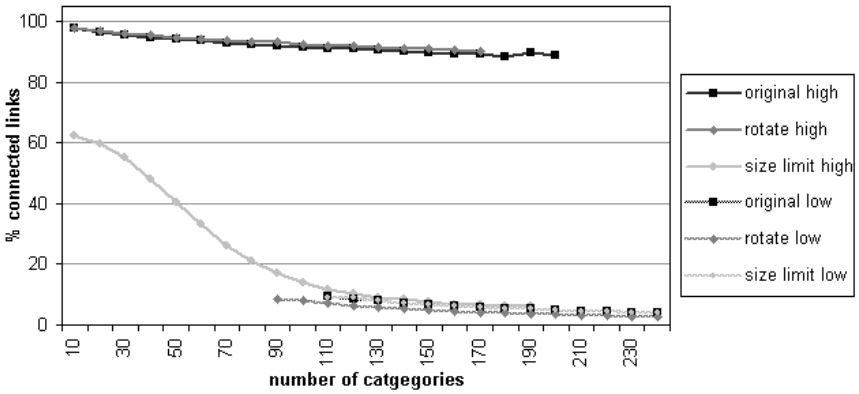

Fig. 4. average % connected links: 100 trials per category.

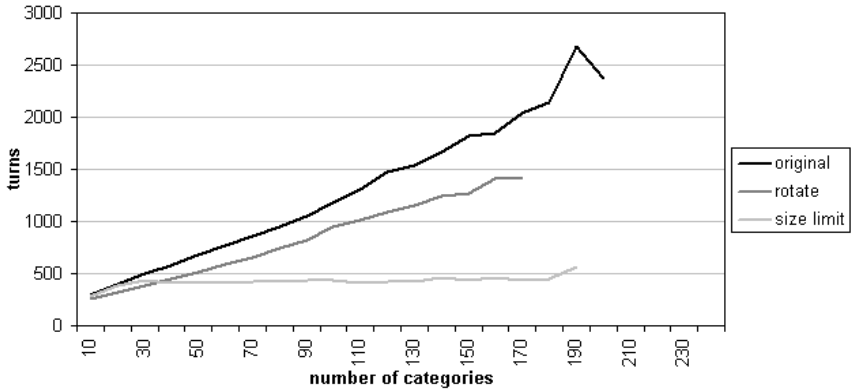

Fig. 5. average number of turns in a connecting trial vs. number of categories: 100 trials per category.

The "size limit" data in Figures 3, 4 and 5 shows results obtained when cluster size is limited by preventing new connections forming once a cluster has reached a given size. The data shown is for clusters limited to 30 agents. Limiting clusters to between 10 and 100 agents provides similar results. The proportions of connected tasks for 100 agent clusters are between one and seven percentage points higher than those shown for clusters of size 30. When cluster size is limited the difference between trials that connect and those that do not is less clear-cut. We separated trials into those that formed a maximum sized cluster and those that did not. Figure 3 shows that the size limited system supports the same number of categories. However Figure 4 shows that as the number of categories increases the percentage of connected links in a trial that forms a "large" cluster decreases towards that of trials that do not connect. We however find in the next

section that this limitation does not make as much of a difference in a dynamic system.

5.3 Task completion

We now extend our model to cover a dynamic system in which tasks are completed and agents then search for partners for new tasks. We add to the system's parameters a probability, P_e, of a connected client-server pair's task ending at a given turn. This represents different jobs that take different amounts of time to complete. When a task between two agents completes, each agent is given a new task of a randomly selected category. The link between these two new tasks is left in place. The cluster center checks if breaking this link breaks up the cluster and if so forms two separate clusters.

Figure 6 shows sample trials with 60 categories, a cluster size limit of 30 and various values of P_e. Things go well for a time, but the number of connections in the system gradually decays and eventually the system falls apart. Figure 7, which shows the category distribution at the start, middle and end of the run with $P_e = .003$, shows why. In this graph client-server pairs are setup so c_0 matches c_1, c_2 matches c_3 etc. At the start of the run the distribution of tasks is pretty much even. At the end however there is a large difference between the numbers of clients and servers in some pairs, thus the number of possible matches in the system is greatly reduced. This happens because matches are made then removed from the system, and eventually a random distribution with a large enough number of members will create a large discrepancy between categories. Since the system needs a minimum number of possible matches, as shown in previous experiments, it will always eventually decay. However, a system where category c_i matches itself avoids this problem and will continue to form connections indefinitely.

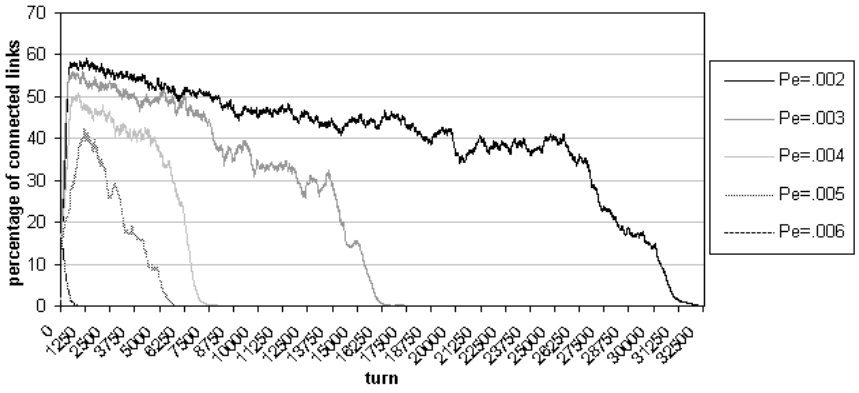

Fig. 6. % of connected links over time when tasks end: 60 categories, size limit 30.

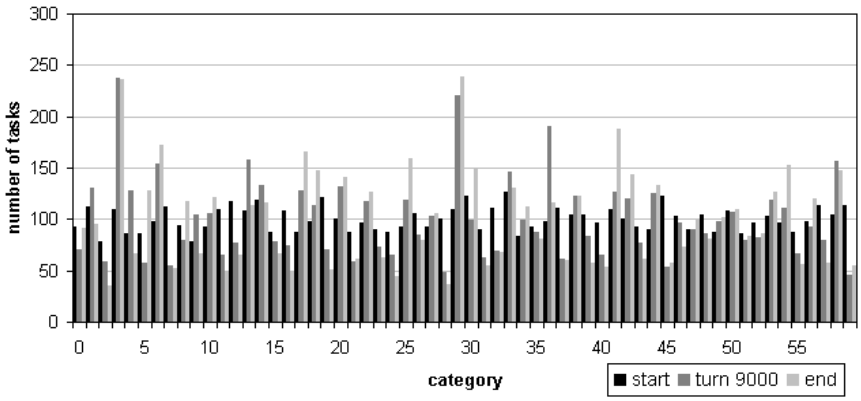

Fig. 7. category distribution for the P_e=.003 run in Figure 6.

5.4 Task completion with flexibility

To combat this difference in the number of clients and servers we must add a new element to the system. We create a new parameter P_c, the probability that an unmatched task will change to a new type at each turn. This represents agents giving up on tasks when they cannot find a match for them. Figure 8, which shows sample runs with various P_c values shows that with this mechanism the system no longer decays. Figure 9 shows a longer run for $P_c = .00005$. This run is interesting since it shows a system where clusters form, change, and occasionally completely break apart then form again. Note that in the static case a size limit of 30 produced trials that only formed about 33% of connections, whereas in the dynamic cased they can achieve almost 55% connected links. In a dynamic setting the number of connections at any one time however is not as important. Looking at the tasks completed per agent over time we find that all agents complete comparable numbers of tasks.

5.5 Unsynchronized turns

Turns in the experiments shown above are synchronized; during each turn each cluster is allowed to move once. Truly decentralized systems do not have such a global clock, thus our experiments assume that all clusters move at the same speed. One way of simulating an unsynchronized system is to move clusters in a randomly chosen order. We repeated the above experiments moving one cluster chosen randomly from all the clusters at a time. For comparison's sake we marked turns as ending after N moves, where N is the number of clusters at the start of the turn. Results for these experiments showed that such an unsynchronized system supported the same number of categories as our original system with less than 1% difference in the number of turns. Sample runs with tasks ending and changing also came out the same as those shown above.

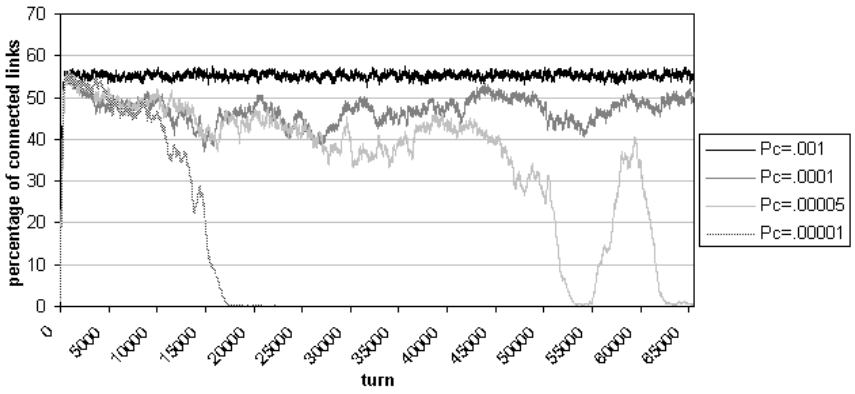

Fig. 8. percentage of connected links over time when tasks end and unmatched tasks change: 60 categories, size limit 30, P_e=.003.

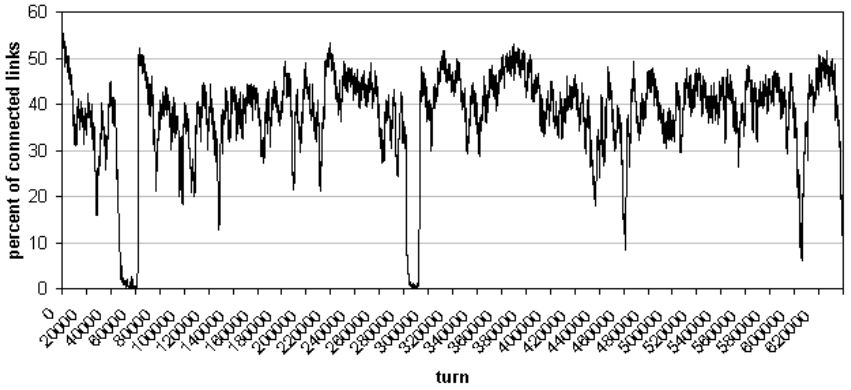

Fig. 9. P_c=.00005, 60 categories, size limit 30, P_e=.003.

6 Conclusion

In this paper we showed that a system of un-coordinated agents is able to solve a matchmaking task under certain conditions. We explored a system where agents look for any one of a number of identical matches. Agents searched among a local set of neighbor agents for these matches, and formed cluster partnerships to expand their search space. We showed that such a system could operate if the size of a cluster is limited to as few as 30 agents. We further investigated a dynamic version of this system where client server pairs completed tasks and went on to search for partners for new tasks. We showed that such systems decay for client-server matches because of an eventual uneven client server distribution. We overcame this problem by adding a chance of agents abandoning unmatched tasks. We showed that this probability could be low, .005% per turn per task, and still allow the system to continue indefinitely.

Our abstract model is however still far removed from a real system. Currently the categories of tasks assigned to agents, how long tasks take to complete and the chance of an unmatched task being reassigned are independent random variables. In future work we would like to study systems where these values depend upon the current and past values of other tasks in an agent. This would represent agents carrying out planned series of actions. We would like to know if in such a system all agents are able to complete their programs, or if some will be in positions to perform better than others.

References

1. Awerbuch, B., Peleg, D.: Online Tracking of Mobile Users. Jounal of the ACM, 42(5), (1995) 1021-1058
2. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. Proceedings of the 15th International Joint Conference on Artificial Intelligence. (1997)578-583
3. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, English ed., Addison-Wesley, Edinburgh, (1999)
4. Kuokka, D., Harada, L.: Matchmaking for Information Agents. Proceedings of the 14th International Joint Conference on Artificial Intelligence. (1995) 672-678
5. Mullender, S., Vitányi, P.: Distributed MatchMaking. Algorithmica, 3, (1988) 367-391
6. Ogston, E.,Vassiliadis, S.: Matchmaking Among Minimal Agents Without a Facilitator. Proceedings of the 5th International Conference on Autonomous Agents.(2001)608-615
7. Shehory, O,: A Scalable Agent Location Mechanism. Lecture Notes in Artificial Intelligence, Intelligent Agents VI, M. Wooldridge and Y. Lesperance (Eds.). (1999) 162-17
8. Shehory, O., Kraus, S.: Methods for Task Allocation via Agent Coalition Formation. Artificial Intelligence, 101(1-2), (1998) 165-200
9. Smith, R.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions On Computers, 29(12). (1980) 1104-1113
10. Sycara, K., Lu, J., Klusch, M., Widoff, S.: Matchmaking among Heterogeneous Agents on the Internet. Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace. (1999)
11. Vulkan, N., Jennings, N.: Efficient Mechanisms for the Supply of Services in Multi-Agent Environments. International Journal of Decision Support Systems, 28(1-2) (2000) 5-19