Matchmaking Among Minimal Agents Without a Facilitator

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

Multi-Agent Systems are a promising way of dealing with large complex problems. However, it is not yet clear just how much complexity or pre-existing structure individual agents must have to allow them to work together effectively. In this paper, we ask to what extent agents with minimal resources, local communication and without a directory service can solve a consumer-provider matchmaking problem. We are interested in finding a solution that is massively scalable and can be used with resource poor agents in an open system. We create a model involving random search and a grouping procedure. Through simulation of this model, we show that peer-to-peer communication in a environment with multiple copies of randomly distributed like clients and providers is sufficient for most agents to discover the service consumers or providers they need to complete tasks. We simulate systems with between 500 and 32,000 agents, between 10 and 2000 categories of services, and with three to six services required by each agent. We show that, for instance, in a system with 80 service categories and 2000 agents, each requiring three random services between 93% and 97% of possible matches are discovered. Such a system can work with at least 90 different service categories and tens of thousands of agents.

Keywords

Agent architectures, multi-agent collaboration, multi-agent simulation, middle agents, matchmaking

1. INTRODUCTION

How do we get computers to solve large, complex problems? The traditional answer is conceptually straightforward - write large complex programs. On the other hand, many problems are too intricate, ambiguous, widely distributed, or dynamic for humans to handle in this way. *Multi-agent systems* present an alternative approach for dealing with complexity. In this approach, in place of simplifying a problem by procedural decomposition one instead programs a collection of agents with small sub-problems. These agents can then *themselves* work out how to compose their capabilities to achieve larger goals. There are many differing schools of thought on how to approach the design of multi-agent systems. The approaches diverge along two main dimensions: the complexity of the individual agents and the amount of centralized control or global knowledge in the overall system. Most approaches attempt to add some structure, or to put some high level programming into the individual agents to get them to cooperate, communicate, or have some sort of goal-following behavior [2] [5].

In this work, we concentrate on reactive agents in systems without any pre-existing external structure. We do this in order to minimize system complexity and the individual agents' resource requirements. We attempt to discover if there are conditions under which it is possible for coordination to emerge naturally from an unstructured initial configuration of trivially simple agents. To carry out our study we have chosen to consider matchmaking, a task common to any system where subtasks are distributed and organizations are not fixed. The matchmaking problem asks how agents providing a service can find the agents who wish to use that service, and vice versa.

Many existing agent systems use *middle agents* or *facilitators*, agents that provide a central directory, for task allocation [6] [11]. Such facilitators are costly to build, they must track information on all the providers or consumers in a system, and must be able to handle the communication and processing necessary for large numbers of requests for matches. An alternative is to use market-bidding mechanisms [10] [12]. This approach involves broadcasting bids and offers to all agents in the market and thus entails high communications costs. Such facilitator or market-based systems have a design goal of allowing a consumer to find the best possible provider for its required service out of the entire agent system. We relax this constraint and instead consider a problem where each consumer is merely looking for any one of a number of possible provider matches. This allows us to use peer-to-peer communication between agents, with a number of advantages, including the following:

- The set up cost for the system and the memory, processing, and communications resource requirements of individual agents are kept to a minimum.

- The amount of information stored at, and communication to any one place in the system is minimized. This maximizes the amount of parallel communication possible.

- By avoiding a central facilitator that must understand advertisements for services, the need for a single common system wide (capability) language is reduced.

- Privacy concerns involved with advertising requests or capability information to the entire system are reduced.

This paper presents a parameterized model of such a peer-to-peer architecture. Each agent in the model has a number of tasks for which it must find help to complete. These agents randomly search among their neighbors for matching abilities. When a match is found the two agents involved are considered able to cooperate. They are combined into a cluster that allows each agent to extend its neighborhood and thus the search space for its remaining tasks. This model is analyzed for particular values of the parameters, and robustness as the parameters change. We find that:

- For a range of parameters the system converges to an almost fully matched configuration. With 80 service categories and 2000 agents, each looking for 3 random services 93% to 97% of possible matches are found.

- Under conditions when the system does converge, it does so rapidly and consistently over a class of random starting conditions. For the above parameters convergence occurs within 500 to 900 turns.

- The system can find matches when there are up to 90 categories of services, and preliminary simulations indicate that it could support thousands of categories, depending on the number of tasks each agent is given.

- The time needed to reach an end configuration increases approximately linearly with the number of categories of services and less then linearly with the number of agents. We are able to simulate systems with up to 32,000 agents.

This paper is organized as follows: In section 2 we define more precisely the problem we are exploring and the form of solution we are looking for. Section 3 discusses other approaches found in the literature. In Section 4 we define the model we are working with. Section 5 then presents simulation results and attempts to provide some explanation of how the model behaves. We conclude in Section 6 with a summary of our model's limitations that must be explored in future work.

2. PROBLEM DEFINITION

The problem we are working on is that of Distributed Matching of Service Providers and Consumers. The problem is one of a world where there are a number of services, and a number of providers and consumers for each service. The number of services is potentially very large. Each agent within the system is a provider for some number of services, and/or a consumer of some others. A successful resolution of this problem is one where a large proportion of the agents in the system have their provider and consumer needs satisfied. The solution to such a problem involves the creation of a large number of provider/consumer pairs, formed by a provider and a consumer of a particular service meeting and agreeing to work together. Such a solution can be viewed as coordination between the agents making up the overall system. Rather than trying to write complex agents which reason about the space, and have complicated algorithms for attempting to find other agents with whom they can co-operate, we aim to create a system with the following design goals:

- The agents are very simple. We want to avoid agents that attempt to reason about their situation since these are difficult, if not impossible, to program, and require large amounts of computing resource per agent.

- There is no centralized control. A centralized controller, limits the scaling capability of the system. As the system grows, a central controller that stores data on all agents will need more and more memory. It will also need an increasing amount of processing time to search for matches among its stored advertisements, and it will need more and more bandwidth to handle requests from the agents.

- There is no global knowledge or global structure. Agents have only local knowledge, and there is no entity in the system other than the agents. Global knowledge and structure can be expensive to maintain. They also makes it hard to add new agents which must be initialized somehow.

- The communication requirement between agents is limited. Large amounts of communication to or from specific points can also limit the scalability of the system and place high resource demands on individual agents. Moreover, we do not want every agent to be required to speak a language that would allow it to communicate with any other agent in the system. Such a requirement would be a form of global knowledge that would make it more difficult to add new agents.

If a solution can be found under such constraints, we have for all practical purposes an inherently scalable and open solution to the problem. In the following sections we propose a model based on these guidelines, and run simulations to explore it's behavior.

3. RELATED WORK

There are a number of common approaches to matchmaking involving either broadcast queries or central controllers. In this section, we consider solutions that have been used in multi-agent systems, and discuss some of their advantages and limitations. A more detailed overview of this area is given in Ferber, chapter 7 [2].

The simplest solution is for the system's creator to predetermine how agents will interact. Tasks are decomposed and agents are matched by hand. This is the approach taken in object oriented design and does not strictly fit with multi-agent systems philosophy. However, a multi-agent system with a limited number of agent types and interactions can be designed in this way.

A more flexible approach is the use of markets for task distribution, as studied by Smith in Contract Net [10]. There is a significant amount of research dedicated to market mechanisms and how they can be used to fairly distribute goods. This is easily extendable to trading services in place of goods [12]. Such research focuses on one-to-many or many-to-many negotiation. Markets involve all participants seeing all bids and offers, and thus are well suited to problems that require negotiation to obtain a fair price for services or goods. The market mechanism however depends on a marketplace, which is often centralized. All agents in the market must know bids and offers, and this must be done though a large amount of broadcast communication. The central market controller must keep a list of all the agents participating, and each bid or offer involves sending a message to each participant. Market rounds can involve all agents making a bid or an offer. This can result in a prohibitive amount of traffic in large markets since each agent receives as many messages each round, as there are agents in the market.

A second popular approach is to use a facilitator, such as a broker, to centrally determine matches [6]. There are a number of different versions of facilitators, differing in whether consumer or provider information is stored, and if the facilitator directly connects consumers and providers or acts as an intermediary for transactions [1]. However, all facilitator architectures involve offered services or requests being stored in a central location. Agents wishing to find matches then apply to this location. Facilitator architectures are good for finding optimal matches since the facilitator is in a position to compare all available possibilities. It can then return the agent that best fulfils requirements or best distributes processing load. However, since facilitators provide complete directories of the consumers or providers in a system they can consume a large amount of memory and all advertisements and requests must be made in a way the facilitator can understand. Also since all matches are made through requests to the facilitator they can become a communication bottleneck. To lessen this problem facilitators can be distributed. Mullender and Vitányi [7] present a general model of a distributed directory service and its memory and messaging costs. Jha et al [4] discuss splitting a single facilitator's function among a number of agents.

A third way agents can locate potential partners is by asking each other for recommendations. In acquaintance networks [2] [8] agents in a fixed network get neighbors to pass on requests for services. Thus, each agent only needs to know who its neighbors are. However, in terms of the number of message passes, this is even more costly then directly broadcasting requests. In addition it requires agents to remember which messages they have seen before, to prevent messages being passed round in circles. Forner [3] discusses using recommendations among agents that naturally group into categories to find other like-minded agents. He however encounters a bootstrapping problem; agents still need to find an agent that can provide the service of recommending initial agents to form seed groups. He falls back on broadcast requests or a central registry to solve this problem.

Another way of using acquaintances to allocate tasks is to form coalitions. However choosing which coalitions to form can be difficult. Shehory and Kraus [9] present a distributed algorithm for coalition formation. Still, it involves recursively calculating possible coalition values, and then picking the coalition with the best system wide value. Coalition values can be calculated in parallel, but this phase requires each agent to know of all other agents in the system. Moreover, to determine the best value they resort to broadcast.

In this paper we consider systems where there are multiple possible provider matches for each consumer. Consumers look only for an acceptable match, not the best possible match. Providers and consumers are distributed randomly around the system. Thus, it is perceived as likely that any particular consumer can find the provider he is looking for within a local area. To make use of this property, we have agents search locally for matches, then create simplified coalitions to find further matches. At any point in time an agent knows the location of only as many other agents as it has open matches to find. It queries these other agents directly to determine if they are good matches. If any are acceptable, a match is made and the involved agents form a group. Addresses that do not provide good matches are then exchanged with other group members, and a new set of direct queries are made. Some of the benefits this approach provides are:

- The amount of information stored at any one place in the system is minimized. Each agent only has to keep track of as many other agents as it has open tasks. In our experiments group membership is stored centrally for each group, however this only involves keeping a list of addresses of unmatched tasks, it is not necessary to keep full descriptions of the tasks.
- Communication time is minimized. Queries to check for good matches are made point-to-point between agents, and thus can be done in parallel.
- Consumers directly query their potential providers so only suitable providers need to be able to understand their queries.
- Direct queries allow agents to control who they give sensitive information to.

4. SYSTEM DEFINITION

As stated above, we are looking for solutions to the matchmaking problem that can be applied in an extremely large open system. We assume a system where agents already have a way of knowing what services they are asking for, including knowing what they will give in return. We want to avoid the use of broadcast queries, or of a central registry of available services. We begin by building the simplest model possible of such a system. In this model, our aim is to obtain as fine a grain distribution as possible, and to keep the individual behavior and resource requirements as small as possible. The components that such a model must include are:

- A collection of agents.
- For each agent, a set of tasks that the agent needs to obtain outside help on to complete.
- A way of checking whether two agents are compatible; i.e. if they can cooperate on a particular task.
- A search algorithm with which agents can locate potentially compatible agents.

We instantiate these as follows:

- The overall system has tasks of varying categories, $C = \{c_1, \ldots, c_m\}$, to complete. The number of different task categories, m, is one of the key parameters affecting the behavior of the system. We mostly consider values of m of the order of one to two hundred. We consider briefly systems with up to 2000 task categories.

- The system has a set of n agents: $A = \{a_1, \ldots, a_n\}$. The number of agents is an important parameter, but our goal is to build a system where this parameter is less important than it might be. In particular, in an ideal world, the overall performance of the system should be independent of this parameter.

- Each agent has a set k of tasks, $T_a = \{t_1, \ldots, t_k\}$, it has to find matches for, each task belonging to a category in C. (T_a can contain more than one task from a category) These are assigned at random to represent agents with different overall goals. The overall behavior of the system when this number of tasks is one or two is relatively simple (see below). Interesting behavior arises when each agent has three or more tasks to complete.

- We start with a fixed and trivially simple model of compatibility checking: we ask the question "is this task of this agent-number-one compatible with this task of this agent-number-two" and get a yes/no answer. Thus we define a pairing among the categories: $f : C \times C \to \{0, 1\}$ 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 matches are symmetric.

- Agents start with their tasks paired at random to a task of another agent. This represents an initial uncoordinated state for the system. Such an initial state is derived from some starting connections formed by some means outside of the system, for instance based on location. Agents then do a local random search of the tasks of their neighbors to find matching tasks.

The final element of the model, the way that agents search for compatible partners, needs some more careful description. We want to create a search mechanism that is as simple as possible and to uses only local knowledge. By local knowledge we mean knowledge obtained from agents who are direct neighbors. However, if agents only ever see their neighbors the system will quickly stagnate. We thus allow agents to extend their search space by forming further searching agreements between agents who find that they are compatible when working together on a task. We justify this by considering that agents that can cooperate on a task are likely to have more in common, for instance they might represent devices made by the same manufacture or conforming to the same standard. A high-level description is as follows:

- We consider each task that an agent has to perform as an "interface" to the agent. We visualize the agents being distributed spatially so that for each agent each interface is adjacent to an interface for another agent. This creates a neighborhood for each agent as large as its number of tasks.

- Initially we simply check compatibility amongst adjacent interfaces. If two adjacent interfaces are compatible, we form a "cluster" of the agents, joined along the compatible interface.

- We then allow clusters (individual agents are clusters of size 1) to "rotate", permuting which yet unattached interfaces are paired to which neighboring cluster's interfaces. In the following experiments we do a different random permutation each turn. This enables agents to "search" for a compatible interface among the free interfaces of their neighbors. Two interfaces that are paired up but are not compatible are left unchanged and swapped again on the next turn. When two newly paired interfaces are compatible their clusters are merged. All the unattached interfaces in the resulting cluster become potentially adjacent to any of the adjacent interface to the original clusters.

- This is how the search spreads out from adjacent agents. As agents find compatible neighbors, clusters form and (in a successful system) gradually get larger. A successful trial of the system is one that ends up with most of the agents connected to each other, usually in a single large cluster. Note that our success criterion is that *most* of the opportunities for coordination are exploited, not that all of them are. This criterion should be sufficient for numerous practical applications assuming that "most" is in the range of 90% or so.

Thus, we have created a model with the following parameters: the number of categories of tasks within the overall system, the number of agents, and the number of tasks (or interfaces) per agent. Currently, we have fixed an initial value for several other potential parameters. The task distribution among agents is random, and each agent has the same number of tasks. The initial connections among agents are also random and take no account of agent locality. We use a local random search mechanism. We enlarge search spaces by forming search agreements between agents that proved compatible on the task-matching front.

It must be noted that in this model cluster size is not limited and thus our design goal of avoiding centralized control is not fully met. A cluster can grow to include almost all agents in the system, and in our simulations show a cluster can have up to of 1/3 of the interfaces in the system to keep track of. This problem of limiting cluster size or distributing cluster operations is left to be addressed in later work. For now we concentrate on the interesting behavior the simpler system displays. We also note that there are two ways of defining our pairing function f. Categories can either each match to themselves, or can form client-server pairs. In this paper we consider the first case, however simulations confirm that there is no noticeable difference between the two. In each case any two randomly chosen interfaces still have the same chance of being compatible.

5. RESULTS

The above model was coded into a simulation tool that allows us run hundreds of trials of experiments to gather statistical data. The following section describes our exploration of system behavior through these simulations. In this section, we first describe what happens with an initial set of parameters and then vary each of the parameters to get a broader picture.

5.1 Basic behavior

We begin by investigating systems of agents with with three interfaces each. Three interfaces should be the lowest interesting number, agents with one interface will at best form pairs, and agents with two interfaces lines and circles, neither creating large clusters.

We first consider an extremely low number of categories of tasks. Figure 1 shows a trial run with 10 categories and 2000 agents. The Y-axis shows the percentage of interfaces that have found matches, and the X-axis shows time, measured in turns. A turn in our simulation is a period during which each agent or cluster in the system is allowed to move (i.e. shuffle its free interfaces) once. In Figure 1 you can see that with this low number of categories almost all the interfaces quickly find matches. We do not expect all of the interfaces to be matched at the end of the run since we assign categories at random, and thus there can be an odd number of interfaces of a particular category.

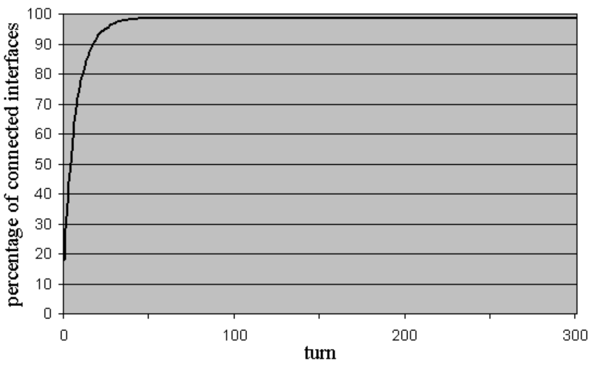

Figure 1: 3 interfaces, 10 categories, 2000 agents

Figure 2 shows the other extreme, a trial with 300 categories of tasks. Here the chance of interfaces finding a match is so low that only a few tiny clusters form. Some agents are lucky enough to be near to matching agents in the initial set-up. However, after these matches are made, the agents find themselves surrounded by others with whom they have no tasks in common, and no further connections are formed.

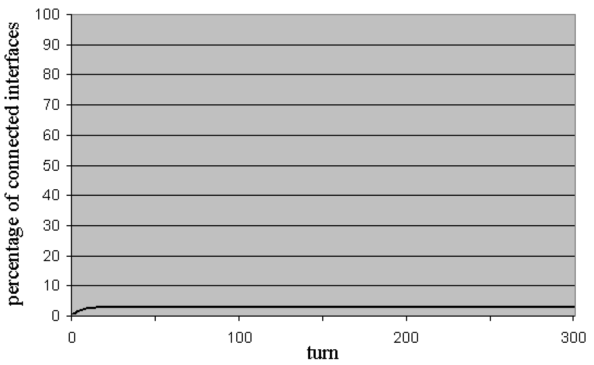

Figure 2: 3 interfaces, 300 categories, 2000 agents

We now consider a system with an intermediate number of categories of tasks . Figure 3 shows a run with 80 categories. We again create 2000 agents, giving an average of 75 interfaces of each category, enough that the chance of having a very uneven distribution of categories is low. What we intuitively expect to happen is that some of the interfaces will start next to matches, thus forming many small initial clusters. Each turn some number more will find matches so that these clusters will grow. Two randomly chosen interfaces have a 1/80 chance of matching, so clusters will probably grow steadily and slowly, and eventually reach a point where the available search space for each cluster contains no more matches. At this point, the system will stop changing.

This intuition however fails to materialize. There are a couple surprises depicted in Figure 3; first the fact that 95% of the interfaces find matches, and second, the shape of the curve. After a set of initial connections, matches are found slowly as we expected, but then suddenly the system takes off and the remaining interfaces are matched up at a much faster rate. This happens over all the random starting conditions we generate. The turning point varies slightly but not much.

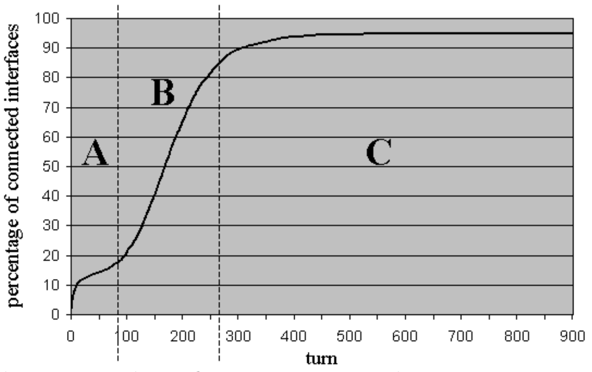

Figure 3: 3 interfaces, 80 categories, 2000 agents

A closer inspection of the size of the clusters on each turn gives us a better intuition into this behavior. This is hard to depict in a graph, so instead we will give a description. The initial set-up creates many small clusters as it did in Figure 2 when there were a large number of categories. Each of these clusters grows slowly during the first part of the curve, region A in Figure 3. However, once one of these clusters becomes large, it will have many free interfaces. The search space of each of these interfaces and thus their chance of forming a connection grows. Moreover, when a new connection is formed the chance that it is to another cluster instead of to an individual agent becomes large. Once this happens the cluster grows even bigger, and its chances of connecting to other clusters are even larger. In regions B and C of Figure 3 the system acts more like Figure 1, the 10 categories case. In region B, there is a phase where all the small clusters are rapidly grouped together into one large cluster. This cluster has many free interfaces, which quickly connect to the remaining free agents in the system. The rate at which connections form then slows down; new connections are between agents already within the cluster. In region C of Figure 3, the agents are mostly interconnected and the search space for each free interface is all of the other free interfaces. Thus in this region we see a slow down in connection rate as the remaining free interfaces in the final cluster pair up.

In these simulations, we have a centralized controller for each cluster that shuffles the free interfaces and reassigns pairings sequentially. If this process is not much faster than the following parallel check for task compatibility, a turn for a cluster with many free interfaces will take longer than a turn for a cluster with few free interfaces. We do not show this on our graphs as we anticipate creating a version of the system with decentralized clusters.

5.2 Number of categories of tasks

In the previous section we saw that our system has two basic kinds of behavior; with 300 categories of tasks the agents remain separate and almost no interfaces are matched, while with 10 categories agents connect into a single large cluster with almost all interfaces finding matches. However, what happens in between? We looked at one point, 80 categories, between these two extremes and found it acted most like the 10 categories case. However, how does this behavior change as the number of categories is varied? Does the percentage of connected interfaces at the end of the trial slowly diminish until it reaches the 300 categories case? Alternatively, is there a region where some trials form a large cluster as in Figure 1 and some behave as in Figure 2? How rapidly does this change occur?

Further experimentation shows that all trials, no matter what the parameters, either form more than 90% of possible connections or don't connect at all, forming less than 15% of possible connections. Figure 4 shows what happens as you change the number of categories. The X-axis is the number of task categories in the experiments run; the Y-axis shows the percentage of trials that form a large cluster. Interestingly, for 90 categories and below trials always connect. From there we find a steep drop off where very quickly all trials never connect, or almost never connect (the tail end of the curve is not as precise as the head).

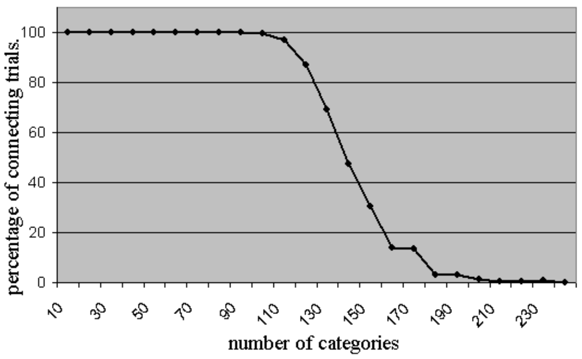

Figure 4: Percentage of trials that formed large clusters; 3 interfaces, 2000 agents, 100 trials per point

This is useful from an engineering standpoint. We can say that if a system has less than 90 task categories we can be sure of it connecting, and above that, it is probably not worth running. Ninety categories is a surprisingly large number, probably high enough for simple applications. We show later that as you increase the number of task matches each agent is searching for the number of categories supported by the system increases. For 6 tasks per agent we can have up to 2000 categories.

We are also interested in knowing how large an effect the number of categories has on running time. Figure 5 shows that as you increase the number of categories the time it takes for the system to connect appears to increase linearly. Here the X-axis shows the number of categories, and the Y-axis shows the number of turns run until no more changes occur in the system. We plot data from 100 trials at each point, showing the maximum, minimum and average values. The upper set of lines represents the trials that formed a large cluster; the lower set of lines represents the trials that remained unconnected.

In Figure 6 we compare some typical connecting runs for

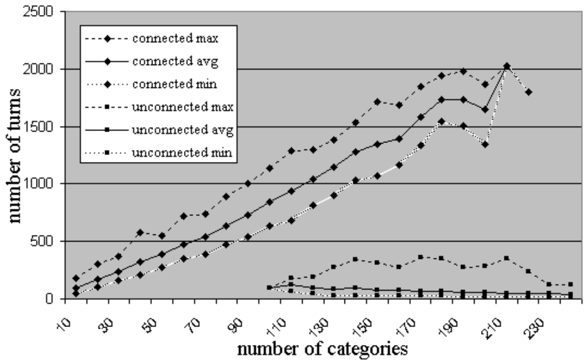

Figure 5: Time until trials stop changing; 3 interfaces, 2000 agents, 100 trials per point

trials with 10, 50, 90, 130 and 170 categories. This shows how the shape of the curve in Figure 3 changes as the number of categories changes. The A region of Figure 3 grows longer and the steep part of the curve in region B becomes more gradual. The maximum percentage of connected edges also decreases, however this effect is lessened for trails with more agents. Trials with 120 categories and 2000 agents form between 92.5% and 94.9% of possible connections, for 20,000 agents this increases to between 97.7% and 98.2%.

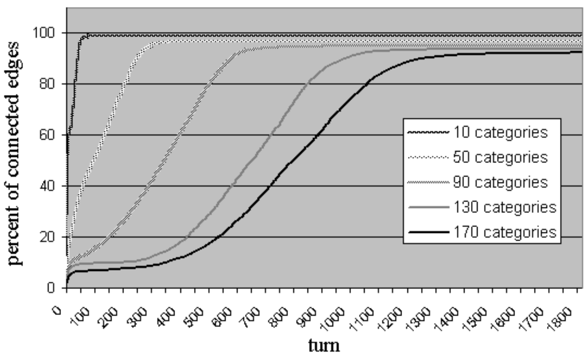

Figure 6: Sample for varying numbers of categories; 3 interfaces, 2000 agents

5.3 Wait time

Looking at the 300 category case in Figure 2, you could speculate that it will eventually behave like Figure 3, the 80 category case, if only we let the system run for longer. This could be especially true when the number of categories is just large enough that the system doesn't always connect. Perhaps waiting longer would allow us to get a higher percentage of connecting trials when running experiments with large numbers of categories.

In Figure 4 we waited 100 turns after the system stopped changing before terminating a run. Running this experiment with a wait time of 400 or 800 turns produces no difference in the results. This is because when clusters are small the search space of their free interfaces is also small. One hundred turns are enough to cover all the possible combinations of pairings. The search space for a cluster only changes if it forms a new connection or one of the clusters near it forms a new connection. If no clusters change during 100 turns none of the search spaces change. This means we can be fairly sure that the system has stagnated. This is a valuable property; it means that an individual cluster has a guideline to determine if it is in a system, or a part of a system, that has come to a standstill. We can use this later when we want agents to be able to determine if they should switch to a more complex behavior to make a system connect.

5.4 Number of agents

We now examine how well the system scales as the number of agents increases . How much more time does it take the interfaces to connect? Does the number of categories that the system can support change?

In Figure 7, we graph the number of categories at which 50 percent of trials connect for experiments with differing numbers of agents. In Figure 4, this occurs at just under 140 categories. We use the 50 percent point rather the point at the end of the 100 percent success rate because it is easier to determine. The X-axis in Figure 7 show $\log_2$ of the number of agents in the system; the Y-axis is the number of categories at which 50 percent of all trials connect. From the graph it appears that the number of categories supported by the system increases approximately logarithmically with the number of agents in the system.

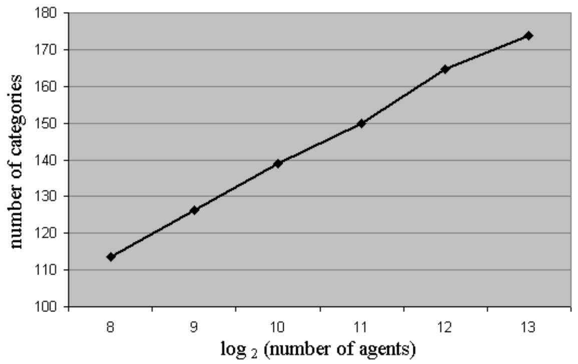

Figure 7: Number of categories for which 50% of trials are successful

In Figure 8 we fix the number of categories at 120 and plot trial length as a function of the number of agents. We can see that this time increases slowly, and perhaps even logarithmically. Based on Figures 7 and 8, we can say that we have a system that scales well with the number of agents.

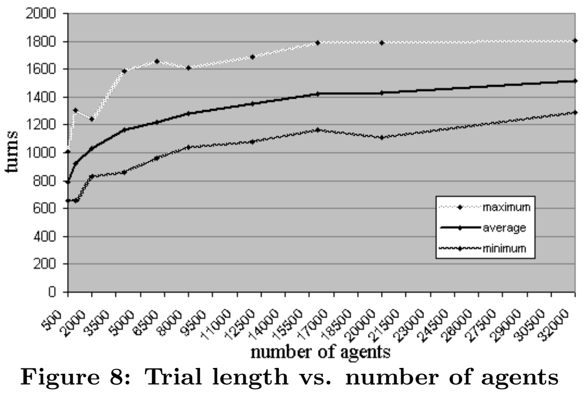

Figure 8: Trial length vs. number of agents

5.5 Number of interfaces

We have done some initial exploratory experiments to examine system behavior as the number of interfaces per agent increases. We find that the number of categories supported by the system increases quite dramatically. This is sketched in Figure 9. We also find that the shapes of the curves shown above for 3 interfaces remain generally the same. However, they become elongated and less precise as the number of interfaces increases. As this research is still in progress, we leave a more thorough exploration of these properties for a later report.

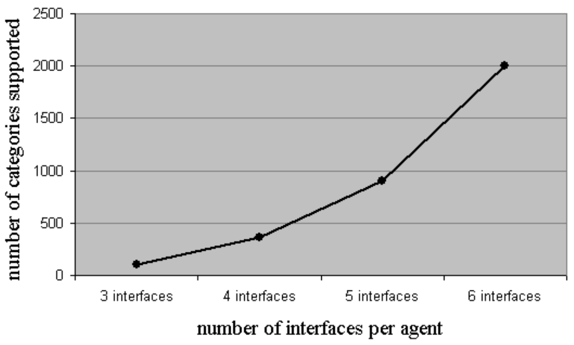

Figure 9: Number of categories supported vs. number of interfaces per agent

6. CONCLUSION

In this paper, we have studied a system of simple agents that search for service consumer-provider matches without resorting to a central facilitator or broadcasting requests. We considered systems where consumers are looking for any one of a number of possible providers, and made use of the fact that a consumer probably dose not have to search the entire system to find an acceptable provider. We attempted to minimize the resources used by individual agents, requiring them to store only the addresses of as many other agents as they had open tasks. We also attempted to minimize the communications time in the system, having agents talk directly to possible matches. This allowed us to check a large number of possible matches in parallel, and removed the need for a system wide capability description language that all the agents could understand.

We have shown that such an agent model is able to solve a distributed matchmaking problem using only local information. We found that the clustering behavior that our agents use to increase their search space gives the system some interesting and useful properties. There is an avalanche effect in the success of agents in finding partners for their tasks. Once some agents are successful in collaborating, they increase their search space, and that of their neighbors, thus increasing the chance of success for themselves and all those around them. This means that once a “seed” set of agents is successful we are guaranteed of the success of almost every agent in the system. It also means that each individual's local chance of success is not solely what determines its success or failure as a whole. Thus a system with very low local probabilities of agents matching tasks with their neighbors can still coordinate, as long as the chance that some agents somewhere will be successful is high.

We have run simulations of this system with up to 32,000 agents, 10 to 2000 categories of tasks, and three to six tasks per agents. We looked in detail at systems with 2000 agents, each searching for three task matches. We found that for these parameters, and when the system contained fewer than 90 task categories, configurations where 90% or more of possible matches were made were found consistently over all the random initial setups generated. The time needed to find these configurations increased linearly with the number of categories, and remained below 1000 turns for 90 categories. We also did some initial experiments involving larger numbers of agents. We showed that more agents could support more categories. Systems with 32,000 agents, each with three tasks, could support around 120 categories, while the time they took to run increased less than linearly. 32,000 agent systems ran in less than 18000 turns. Finally we looked briefly at increasing the number of tasks per agent, showing that this had an even greater effect on the number of categories supported, with six tasks per agent we could have up to 2000 categories.

Future work on this approach to matchmaking must be aimed at decreasing the gap between our skeleton model and a real world system. We intend to continue our research in a number of directions, as described below.

First, we need to discover a way of decentralizing our clusters, or of limiting cluster size. The clusters in our simulations are centralized. A cluster controller deals with the swapping of free interface connections each turn, and does it in a uniform random way. For some applications, this may be an acceptable limitation. For instance, if the time for agents to check for and negotiate matches takes much longer than the time the cluster center takes to reassign partners. However, memory usage still places a limit on how large a cluster can grow. Centralizing clusters also makes splitting up clusters difficult once a task between two agents has completed. It is possible to decentralize part of the cluster center's function within each cluster. We have run simulations that show it is possible to use the same permutation to swap partners each turn, if time is asynchronous. This means that each agent only needs to be told once, each time its cluster changes, where to pass unaccepted partners. However, combining or splitting up clusters without some global cluster information is more difficult. Should we fail to decentralize clusters completely, another alternative would be to limit a cluster's size. However, if clusters are kept too small they could remain separate and limit the systems ability to reach a fully matched configuration.

Second, in our current experiments we only simulate agents finding partners and building up clusters, we do not consider what happens when partnerships end. Our model should be extended to incorporate tasks that have a limited duration, and agents that go on to perform new tasks once they have completed their original ones. We would like to create a system where clusters are continuously built up and broken down again as partnerships are created then completed, and new partnerships are formed. This mechanism should also include adding and removing agents.

Third, we need to refine our model to run in continuous rather than discrete time. Currently, cluster movement is synchronized by the simulator, each cluster moves once each turn. A real system is more likely to be run in continuous time, with some clusters moving faster or more often than others do. In addition, if we remove cluster centers then the clusters as a whole should also be unsynchronized.

Fourth, we need to run simulations with differing distributions of task categories and initial connections. In our model the distribution of categories is uniform, both in the number of tasks of each category, and location of these tasks within the system. In a real system, we are likely to find that certain types of agents and thus certain categories of tasks and combinations of tasks are more common than others. This can change the probability of seed clusters forming, or stop clusters from growing beyond a certain size. In addition, initial connections are unlikely to be completely random in a real system. We need to look at simulations where initial connections take into account possible locations of agents, for instance a system where agents on a single machine are randomly connected, but connections between agents on separate machines are sparser.

Finally, we need to determine how many tasks agents have in real world problems, and how many categories of tasks are typical. We found our system could support at least 90 categories when we gave each agent three tasks, and this grew up to 2000 categories when agents had six tasks. We however did not look at systems where agents had varying numbers of tasks.

7. REFERENCES

[1] Decker, K., K. Sycara, and M. Williamson, "Middle-Agents for the Internet," *Proc. Of the 15th Int. Joint Conference on Artificial Intelligence*, (August, 1997).

[2] Ferber, J., *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, English ed., Addison-Wesley, Edinburgh, (1999).

[3] Forner, L. N., "Yenta: A Multi-Agent, Referral-Based Matchmaking System," *Proc. of the 1st Int. Conference on Autonomous Agents*, 301-307, (February, 1997).

[4] Jha, S., P. Chalasani, O. Shehory, and K. Sycara, "A Formal Treatment of Distributed Matchmaking," *Proc. of the 2nd Int. Conference on Autonomous Agents*, 457-458 (May 1998).

[5] Jennings, N. R., K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1), 7-38, (1998).

[6] Kuokka, D. and L. Harada, "Matchmaking for Information Agents," *Proc. of the 14th Int. Joint Conference on Artificial Intelligence*, 672-678, (August, 1995).

[7] Mullender, S. J. and P. M. B. Vitányi, "Distributed Match-Making," *Algorithmica*, 3, 367-391 (1988).

[8] Shehory, O, "A Scalable Agent Location Mechanism," *Lecture Notes in Artificial Intelligence, Intelligent Agents VI, M. Wooldridge and Y. Lesperance (Eds.)*, 162-17, (1999).

[9] Shehory, O., and S. Kraus, "Methods for Task Allocation via Agent Coalition Formation," *Artificial Intelligence*, 101(1-2), 165-200 (1998).

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

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

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