The Effects of Cooperation on Multiagent Search in Task-Oriented Domains [*]

José M. Vidal
Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
vidal@sc.edu
http://jmvidal.cse.sc.edu

ABSTRACT

We study the benefits of teaming and selflessness when using multiagent search to solve task-oriented problems. We start by presenting a formal framework for multiagent search which, we show, forms a superset of the task-oriented domain, coalition formation, distributed constraint satisfaction, and NK landscape search problems. We focus on task-oriented domain problems and show how the benefits of teaming and selflessness arise in this domain. These experimental results are compared to similar results in the NK domain—from which we import a predictive technique. Namely, we show that better allocations are found when the dynamics of the multiagent system lie between order and chaos. Several other specific findings are presented such as the fact that neither absolute selfishness nor absolute selflessness result in better allocations, and the fact that the formation of small teams usually leads to better allocations.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence, Multiagent Systems*

General Terms

Multiagent Search

1. INTRODUCTION

Multiagent systems are especially suited to solve problems in which individual decision-makers with localized information are able to affect their local state in the hopes that the system will eventually reach a global state of either optimal or at least satisfactory utility. Classic example scenarios include distributed sensor monitoring [2], distributed task allocation [5], and coalition formation [6]. In these problems, each agent perceives some part of the global state and takes actions that modify some part of this state. The agents act to maximize some local utility function. The function's details, e.g., how "selfish" it is, as well as the interaction protocols among the agents are left to the system designer. The designer must engineer these so that locally optimal decisions give rise to the best possible global state. We refer to these types of problems as instances of a more general **multiagent search** problem.

In this paper we first introduce a formal framework for multiagent search that, we show, forms a superset of task-oriented domain, coalition formation, distributed constraint satisfaction, and Kauffman's NK landscapes [3]. The grouping of all these different problems into one framework allows us to leverage results from one domain and use them in another. As an example this power in Section 3 we present our results on the effectiveness of cooperation via team formation and selflessness in task-oriented domains. This approach was inspired by the successful use of "patches" in the search of NK landscapes in a two-dimensional grid instantiation [4].

From our experiments we were able to derive several interesting results. We show how agents that form teams and engage in limited forms of selfless behavior find solutions that are of a higher global utility. We show that the best solutions are found in systems that exhibit dynamics that are at the phase transition between order and chaos. These results lead us to suggest that further study should be devoted to the study of coordination protocols that do not converge to a stable solution but instead continue to change. We believe that such protocols shall result in better (from a global perspective) emergent behaviors in multiagent systems. We also present several specific findings such as the fact that neither absolute selfishness nor absolute selflessness result in better allocations, and the fact that the formation of small teams usually leads to better allocations.

2. MULTIAGENT SEARCH FRAMEWORK

In this section we present a formal framework for describing multiagent search problems. These problems are characterized by a global state composed of the aggregation of the value of many local variables. Each agent perceives the values some of the variables, modifies the value of some of the variables, and receives a utility that depends on the value of some of the variables. By limiting which variables the agents perceive, modify, or derive utility from, we can instantiate

[*]This work was funded by the NSF under grant 0092593.

USC CSCE TR-2002-012. A shorter version appears in Proceedings of the First Autonomous Agents and Multi-Agents Systems Conference (AAMAS) 2002.

various well-known multiagent problem domains.

The global state is denoted by S. It is formed by the union of a fixed set of local variables $\{s_1, s_2, \ldots, s_{|S|}\}$, each one with a finite domain. The set of agents is $A \equiv \{1, 2, \ldots, n\}$, where n is the number of agents. Each agent $i \in A$ has an utility function $u_i : d_i \rightarrow \mathbb{R}$ that provides a mapping between a subset of state variables $d_i \subseteq S$ and a real number. An agent's relationship with its environment is captured by the set of local variables on which its utility *depends*, the set of local variables whose value it *modifies*, and the set of local variables whose value it *views*. Specifically, for agent i we define d_i to be the set of variables upon which its utility depends, m_i is the set of variables which it can modify, and v_i is the set of variables it can view. The agents modify the state of the variables in their respective m_i sets but only if these modifications satisfy the constraints imposed by $P : S \times S \rightarrow \{0, 1\}$. For example, agent i can only change S into S' if $p(S, S') = 1$, where $p \in P$, and the state variables it modifies are in m_i. If a constraint function evaluates to 0 it means that the particular state change is not allowed.

We now define a multiagent search problem as the tuple $\{A, S, U, D, M, V, P\}$ where $A \equiv \{1, 2, \ldots, n\}$ is the set of agents, $G \equiv \{S_1, S_2, \ldots, S_{|G|}\}$ is the set of all possible global states such that $S \in G$, U is the set of all agent utility functions where $u_i \in U$ and $u_i : d_i \rightarrow \mathbb{R}$, $d_i \in D$, $m_i \in M$, $v_i \in V$, and P is the set of constraints, as defined above.

This formalization of multiagent search states the problem but does not provide a solution. The goal of an agent-based software engineer is to implement agent behaviors that will enable the quick discovery of the globally optimal solution. That is, the system should converge to $s^* = \arg_{S \in G} \max \sum_{i \in A} u_i(S)$. A common approach is the use of individual hill-climbing. In it, each agent modifies its local variables m_i to maximize its utility u_i. It is expected that doing so will also increase the sum of everyone's utility. Unfortunately, this approach usually leads to sub-optimal states. In Section 3 we extend this idea by allowing the formation of teams and the use of partially selfless agents and show the benefits of that approach.

2.1 Task-Oriented Domain

The Task-Oriented Domain (TOD) formalization studied by Rosenschein and Zlotkin [5] is an instance of a multiagent search problem. They define a TOD problem as a set of tasks T, a cost function $c : \tau \rightarrow \mathbb{R}$, where $\tau \subseteq T$, and a set of agents A. The tasks are assigned to the agents. Each agent tries to exchange some of its assigned tasks with other willing agents. The authors restrict the agents to act rationally. They define a rational agent as one which only accepts a deal (i.e., a set of task assignments) if its costs are equal or less in the new deal than in the current deal. One example instance of a TOD problem is the Delivery Problem where a set of letters must be delivered by a set of mail-carriers. Each mail-carrier is initially responsible for the delivery of a subset of letters. Each mail-carrier then trades letters with others in order to decrease the length of the route needed to make all his deliveries.

A multiagent search problem can be reduced to a TOD problem by choosing the appropriate mapping. Each task becomes a state variable whose value is the agent that is assigned to carry out that task. The global state then becomes $S = \{s_1, s_2, \ldots, s_{|T|}\}$ where s_t corresponds to task t. Since all tasks can be handled by all agents, we have that $m_i = d_i = S$ for all agents. The value of v_i will depend on the particular solution algorithm used. That is, the TOD formulation does not specify how much the agents know about the current task assignments. If the agents know who is responsible for every task then $v_i = S$, otherwise it might be that v_i changes dynamically. Finally, the agents' limitations in the changes each one can make to the global state are captured by $p(S, S')$ which is 1 when $\forall_{i \in A} c_i(S') \geq c_i(S)$ where $c_i(S)$ is the cost that agent i incurs in global state S (by handling all the tasks assigned to it in that state). The S and S' are also limited to differ by the value of only one variable because only one task can be transferred between agents at each step.

Different instances of TOD problems may have different cost functions, as well as different restrictions on the types of interactions the agents can engage in. These restrictions are represented by different p functions. The designer of a multiagent system for a TOD problem must design the interaction protocols such that the best possible global state is reached in the shortest amount of time. Although we do not believe that there exists a general approach which will solve all TOD problems optimally, we show in Section 3 how teaming can improve the quality of the solution found in TOD multiagent search problems.

2.2 NK Landscapes

Kauffman's NK landscapes [3], originally intended for the study of evolution's search over gene instances, may also be considered a special case of our multiagent search formalization. An NK landscape consists of N binary "genes". The fitness of each gene depends on its state (zero or one) and the state of K other genes. An instantiation of an NK landscape sets N and K to integer values, where $K < N$, and randomly assigns to each gene a fitness function that depends on the state of the gene and the state of K other genes. An NK model can be reduced to multiagent search by setting $m_i = \{s_i\}$, $d_i = s_i \cup \{K \text{ other state variables}\}$, $v_i = d_i$, and $p(S, S') = 1$.

This reduction is especially interesting because it allows us to leverage research on the characterization of NK landscapes and the effectiveness of genetic searches over this space. That is, even though multiagent problems are different from NK landscapes, we can try to map some of the theorems and results in that domain to the more complex multiagent search domains.

2.3 Distributed Constraint Satisfaction

A distributed constraint satisfaction problem, as presented in [10], is defined as a set of n variables $x_1, \ldots, x_n$, where the value of x_i is taken from some domain D_i, and a set of constraints $p_k(x_{k_1}, \ldots, x_{k_j})$ that operate over these variables. The constraints are boolean functions that must evaluate to true for the problem to be solved. Under a typical distributed algorithm, each agent in the system is assigned one variable. The agent is then responsible for setting its variable to a value that does not violate any constraints.

A straight-forward mapping of this problem to our multiagent search framework is possible. We simply map each variable x_i to one of our local variables s_i. Each agent i is assigned one of the variables so that $m_i = s_i$. Each agent i's utility depends on the set of all variables that share a constraint with i's variable. That is, d_i is the set of all variables for which there exists a constraint between that variable and s_i. Similarly, agent i can either view the state of all variables ($v_i = S$) or be limited to those variables that share a constraint with its variable ($v_i = s_i$). Finally, we define u_i to be one when all the constraints that involve s_i are satisfied and zero otherwise.

By translating a distributed constraint satisfaction problem into a multiagent search problem we are implicitly assuming that solutions which violate fewer constraints are better. However, in a strict interpretation of distributed constraint satisfaction all solutions that violate any number of constraints are equally undesirable.

Finally, we point out that the use of cooperation for solving constraint satisfaction problems has been found to be successful [1], although under a different model than the one used here. Those results are in accordance to the results we present for the TOD in Section 3. However, more research is needed in order to bring the two results together under the multiagent search umbrella.

2.4 Coalition Formation

Coalition formation search can also be considered an instance of multiagent search. This should not be a surprise since it has already been shown [7] that coalition formation and task allocation are related problem domains. The reduction is achieved by assigning an agent to each state variable. The domain of the state variables is a number between 1 and n, the number of agents. It represents the coalition that this agent belongs to. We then define $m_i = s_i$, $d_i = A$, $v_i = A$, and $p(S, S') = 1$ for all agents i and states S and S'.

3. TEAMING AND MULTIAGENT SEARCH IN A TOD

Common techniques to speed up multiagent search in task-oriented domains include communication, delegation (e.g., using contract-net [8]), and the use of auctions [9]. These and other grouping methods align an agent's desires with those of a larger team. However, we do not have any a priori evidence to make us believe that teaming will lead to better solutions for the system as a whole. Namely, we do not know whether the best solution, from a global perspective, will be found by selfish agents, by selfless agents, or by agents somewhere in between.

In this section we instantiate our multiagent search formalism in a task-oriented domain an perform a series of experiments that demonstrate the inherent benefits of teaming and that determine the situations in which selfless behavior is better from a global perspective.

3.1 TOD Problem Specification

We set out to study the benefits of cooperation in a TOD problem, as described in Section 2.1, by randomly grouping agents into teams. The teams are non-overlapping and of a fixed and equal size. The team sizes vary from individual teams where each agent is a team to the grand team where all agents belong to the same team. Agents in a team take actions that maximize the team's utility. We define team(i) to be the set of agents in i's team, including i. We then define the utility that agent i receives in global state S as

$$\text{teamUtil}(i, S) = \frac{1}{|\text{team}(i)|} \sum_{j \in \text{team}(i)} u_j(S). \qquad (1)$$

Finally, we also vary the number of tasks that each agent can do. In the standard TOD problem specification all agents are able to do all the tasks. A common variation is to allow agents the ability to perform only a subset of the tasks. This variation simulates problem domains with heterogeneous agents where some tasks can only be done by some agents. We limit the set of tasks an agent can do by modifying m_i. At one extreme every task can be done by only one agent, in which case the task allocation problem is trivial. At the other extreme all the agents are able to do all the tasks thereby expanding the size of the search space. As such, it is very time-consuming to find an optimal solution for this case.

3.2 Search Algorithm

In order to determine the effectiveness of team formation in TOD we developed a simulator that searches the space of possible states S. Figure 1 shows the main loop of our search algorithm. For each run we randomly generate a new cost function and new starting state. Each step in a run consists of first randomly selecting one agent. This agent then determines which is the best action it can take. The available actions to the agent are to either give one of its tasks to another agent or to take one task from another agent. The agent will consider all possibilities and choose the one with the highest team utility for the agent's team. Also, an agent can only give a task to or take a task from another agent if that agent's utility loss is no greater than the *maximum loss* L, a parameter which we vary from zero, for purely selfish agents, to one, for agents that are willing to take whatever deal is offered to them. That is, if the system is on state S then agent i will only accept a new state S' if willingToDo(i, S, S') is true, which we define as

$$\text{willingToDo}(i, S, S') = \text{teamUtil}(i, S) - \text{teamUtil}(i, S') \geq L. \qquad (2)$$

Agents with a maximum loss of zero ($L = 0$) are not willing to accept any deal where their new team utility is less than their current team utility. These are the rational agents from [5]. On the other hand, agents with a maximum loss of one ($L = 1$) are willing to take whatever deal the other agent offers since the maximum utility loss can never be more than one. These agents could be said to be completely selfless.

3.3 Random Landscape Shared Cost Function Results

For the first set of tests we defined one cost function to be shared by all the agents. The cost for doing every task subset is set to be a random value between zero and one. This means that the costs functions are neither additive nor subadditve. The lack of correlation among the costs of similar

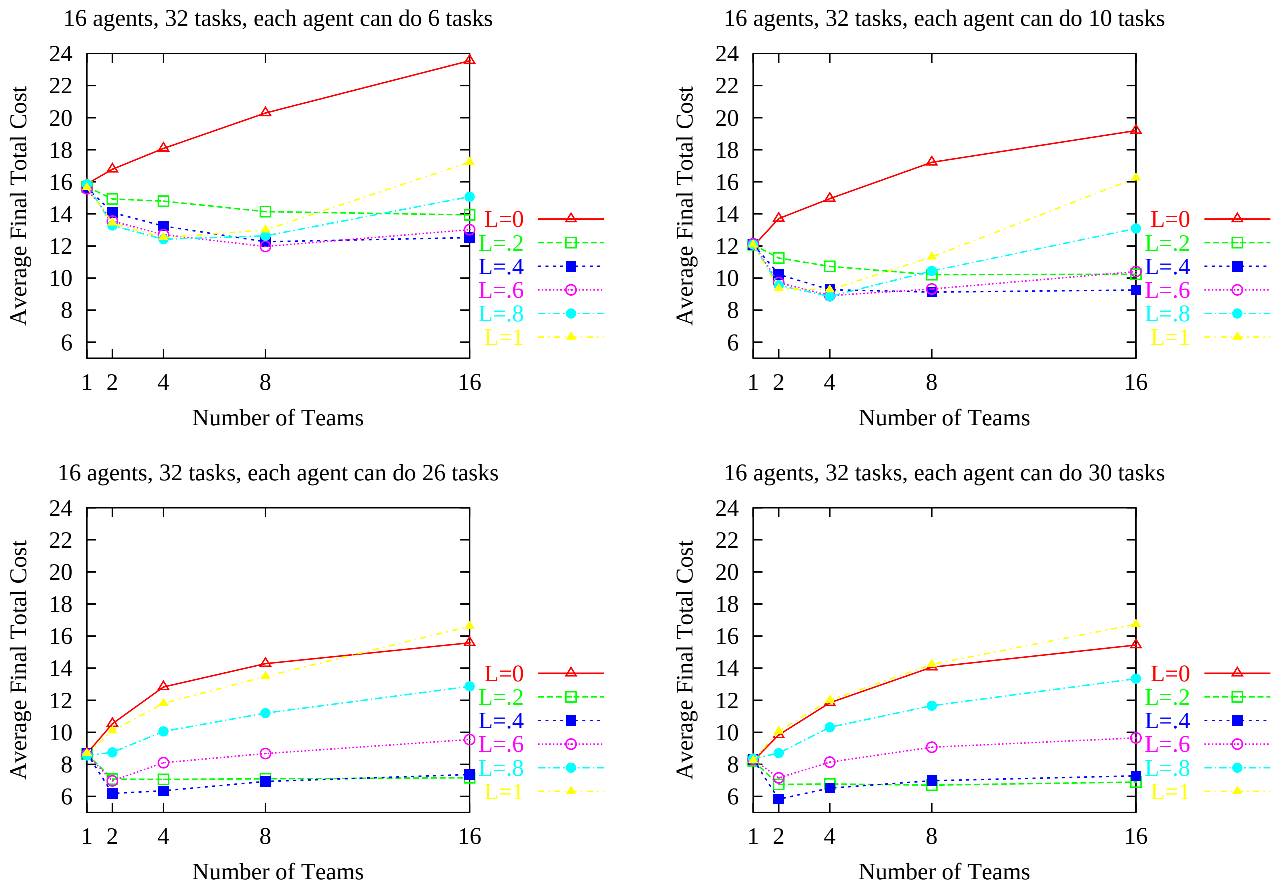


Figure 2: Results for 16 agents and 32 tasks when all agents share the same cost function.

task subsets makes it harder to search this space than if they were correlated. Section 3.4 shows the results when each agent has its own cost function.

Our first experiments involve 16 agents and 32 tasks. For each experiment we changed the number of tasks that each agent can do. Within each experiment we varied the maximum loss parameter (L), as well as the number of teams allowed. These varied from one grand team—everyone on the same team—to 32 individual teams—every agent is in a team by itself. For each combination we executed 1000 runs, each of 500 steps, and plotted the average total cost of the last five states searched. We used numbers of teams that are powers of two so that all teams would be of the same size. We noticed that after 500 steps the cost of the states being visited had usually stabilized so we chose 500 as a suitable number of steps to carry out before stopping the algorithm. The way we have defined our search algorithm it can often keep searching forever, never getting stuck at a local optimum. As such, we had to stop it at some arbitrary point and check the utility of the solution it had found.

The results of our first tests can be seen in Figure 2. It shows the average final total cost for various tests. For each test we ran various populations each with a different maximum loss which is represented with the letter "L" in the figures. The total cost is simply the sum of the costs for all agents. An agent's cost for a particular state is equal the negative of its utility for that state. Figure 3 displays the standard deviation for one of these results, showing that our results are statistically significant. The error bars do not overlap for many of the cases.

Notice how all the curves in the various graphs on Figure 2 have the same value for the case where there is only one team, regardless of the value of L. This is to be expected. Upon examination of our algorithm we notice that an agent that is chosen to act will only pick an S' which has a higher teamUtil than the current state S. As such, it does not matter if the other agent is willing to allow a new state with lower utility since, as both agents are on the same team, the lower utility state will not be chosen by the first agent. The agent will rather stay at S than move to an S' with lower teamUtil. In other words, when all the agents are on the same team the algorithm degenerates into hill-climbing on the global search space. That is, the total cost of S' is always greater than or equal to the total cost of S regardless of L.

The first interesting feature of Figure 2 is the fact that the minimum cost, no matter how many teams are used or how many tasks the agents can do, is always attained by using

```
$S \leftarrow$ Randomly chosen state
for step = 1 to 500 do
  maxUtility $\leftarrow -\infty$
  maxState $\leftarrow S$
  $i \leftarrow$ chooseRandom($A$)
  for $t \in T$ do
    if $s_t = i$ then // $i$ is doing $t$
      for $j \in A - \{i\}$ do
        $S' \leftarrow S$
        $s'_t \leftarrow i$
        if $t \in m_j \wedge$ willingToDo($j, S, S'$) then
          tmp $\leftarrow$ teamUtil($i, S'$) - teamUtil($i, S$)
          if tmp > maxUtility then
            maxUtility $\leftarrow$ tmp
            maxState $\leftarrow S'$
          end if
        end if
      end for
    else // $i$ not doing $t$
      if $s_t = i$ then // $i$ could do $t$
        $j \leftarrow \arg_{j \in A} s_j = t$ // $j$ is doing $t$
        $S' \leftarrow S$
        $s'_i \leftarrow t\}$
        if willingToDo($j, S, S'$) then
          tmp $\leftarrow$ teamUtil($i, S'$) - teamUtil($i, S$)
          if tmp > maxUtility then
            maxUtility $\leftarrow$ tmp
            maxState $\leftarrow S'$
          end if
        end if
      end if
    end if
  end for
  $S \leftarrow$ maxState // Move to best state.
end for
```

Figure 1: The main loop in our TOD search simulations.

a maximum loss value of between .4 and .6. That is, if the agents are either completely selfish ($L = 0$) either as an individual or as a group, or completely selfless ($L = 1$), the group as a whole does not do as well as if the agents are somewhere in between, regardless of the number of teams or abilities of the agents. The most interesting result comes from the fact that the selfish agents ($L = 0$) do the worst of all regardless of the number of teams in almost all cases. This is highly counter-intuitive. One excepts that since selfishness prevents the agent's team utility from ever going down and the global utility is nothing more than the sum of these team utilities, that the team search would be guaranteed to proceed monotonically down in the cost and, therefore, more likely to reach a lower cost for all. However, it seems that allowing the team search to sometimes go up in cost also allows the discovery of even better solutions. On the other hand, if we allow these moves to be too severe, as happens when $L = 1$, then the gains from the extra exploration are lost. Once explained in this way, one immediately recognizes that this is similar to simulated annealing, except that the moves there are completely random. We therefore conclude that partial team-selflessness in multiagent search provides it with some of the benefits of "temperature" in a simulated annealing algorithm, thereby lowering the cost of the final solution.

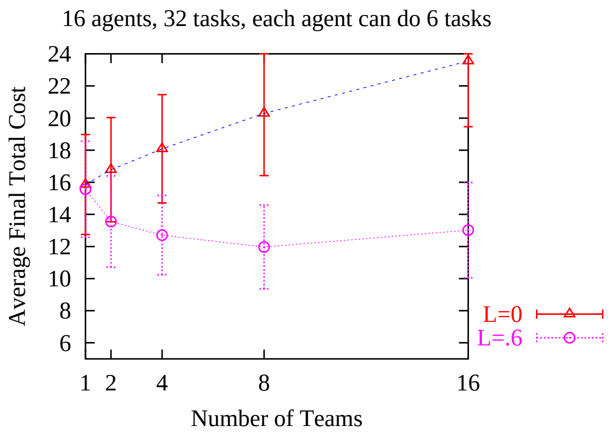


Figure 3: Standard deviation error plot for one case with 16 agents and 32 tasks. The error bars represent on standard deviation.

The second interesting feature of Figure 2 is that it seems like there is no easy way to predict which combination of teams and L values will achieve the minimum total cost. In general it seems like the minimum cost is usually found when using somewhere between two and eight teams. However, this is a large range and we would like a more specific prediction. Fortunately, in [4], Kauffman *et. al.* have studied a similar problem in NK landscapes and found a possible predictor for that domain. Since, as we have shown in Section 2.2, NK landscapes can also be considered an instance of multiagent search we have reason to believe that their results might bear some relevance to ours.

Their experiments consist of agents in a 2-dimensional NK landscape, each agent connected to its four nearest neighbors. The area is divided into square *patches*. Each agent decides whether to flip its state based on the utility that its patch will receive. After carrying out a series of experiments in this domain Kauffman *et. al.* found that the optimal solution is found when the patch size is such that the system's dynamics are between the ordered and the chaotic regimes. That is, large patches make the system quickly converge to a solution state; small patches lead the system into chaotic dynamics where the state is constantly changing. These two diametrically opposite dynamics are also present in our system as we vary the number of teams from 1 to 32. We, therefore, started to suspect that the same phenomena should be present in some way in our domain. Neither we nor [4] can offer a good explanation as to why better solutions are found on systems whose dynamics are at the edge of chaos.

We can show that the best solution is found when the system dynamics are at the edge of chaos by looking at the percentage of runs that converged to a local optimum as a function of the final cost for that run. Figure 4 shows such data for one of the graphs in our first experiment. We notice that, as the number of teams varies from 1 to 16, the sys-

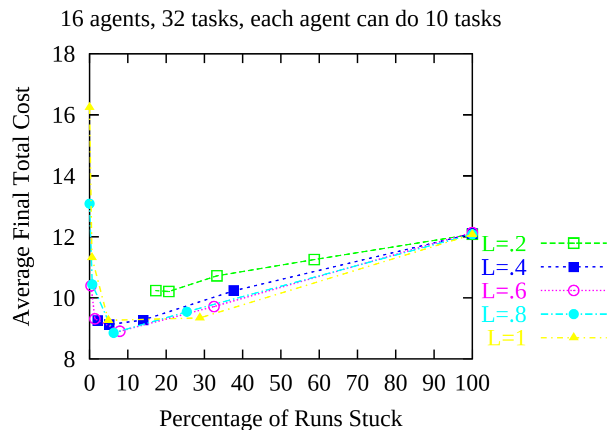


Figure 4: Final cost as a function of the percentage in the number of runs that converged to a fixed point.

tems' dynamics vary from static to chaotic, except for the case where $L = 0$. These dynamics arise because when only one team exists the system is simply doing a hill-climbing search on the global cost landscape. On the other hand, when each agent is its own team it is much more likely that any one global state will not be a local optimum for one of the agents. This agent, if chosen to act, will move that global state away from the otherwise local optimum. As such, it is hard for the system to ever converge to a local optimum.

This result is extremely important because it suggests that the best multiagent search algorithms, from a global utility perspective, are those whose dynamics are at the edge of chaos. Current research in multiagent systems' protocols centers around the idea of reaching an equilibrium where each agent knows what to do and does not have any incentive to change its allocation. This bias comes from the strong influence of game theory and economics on multiagent systems design. However, there are cases where an equilibrium solution will not find the best solution. In fact, as we have shown, in this task-oriented domain the equilibrium solution is guaranteed to be inferior.

It is also interesting to note that our results diverge from [4] in the case where the maximum loss is 0, which would seem to be the case that most closely matches their simulation since their patches never cooperate with each other. That is, their patches act like teams with $L = 0$. An agent in a patch never surrenders some of its utility for the benefit of an agent in another patch. However, their patches overlap so that one agent can be part of two patches. We believe that it is this overlap that changes their dynamics to act more like our $L > 0$ cases. That is, overlapping team memberships seems to have similar effects as selflessness.

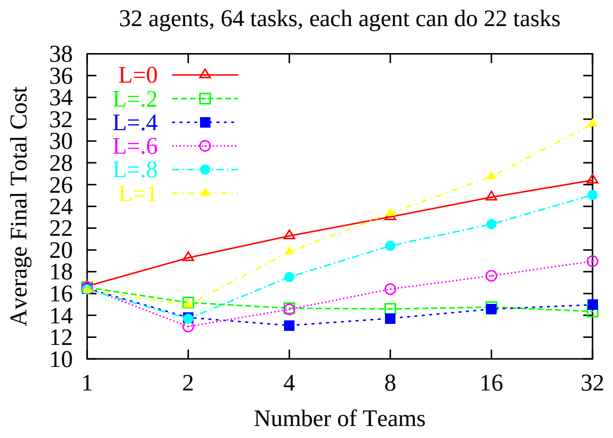
A third interesting and encouraging feature of Figure 2 is the fact that small teams of size one and two can do well, if the maximum loss is adequately chosen. Specifically, the maximum loss needs to be about .4 for this scenario. This result is encouraging because it is these smaller team sizes which more faithfully replicate the physical constraints of most multiagent systems. In many multiagent systems there is little to no communication among agents. In the domain we are simulating, an agent needs to know the state of all the other agents in its team in order to calculate the utility of the team. Therefore, the larger the team the more communication that will be necessary. We conclude that, in general, we should be able to construct effective multiagent systems with small teams and low communications overhead if we allow these teams to act somewhat selflessly.

Similarly, another encouraging feature from Figure 2 may be discerned by drawing on each graph a horizontal line that crosses the point that all curves intersect when the number of teams is zero. Any points that fall below this line are cases where breaking the system into smaller teams actually results in a lowering of the final cost. That is, these are all the cases where subdividing the problem helps the system find a better solution (even without taking into account the added benefits that might accrue from the reduced inter-agent communication). Subdividing the problem clearly helps in the majority of cases. The graphs also show that as agents are able to do more and more tasks the number of points below this line decreases. That is, as the agents become more homogeneous in their abilities to carry out certain tasks the benefits of teaming are reduced. This result has direct implications to the design of multiagent systems.

A fourth feature of Figure 2, one that is related to the previous one, is the fact that the curves with $L = 1$ show better values for the cases where the agents are able to do fewer tasks. That is, acting selflessly ($L = 1$) helps the system reach a better solution when the agents are heterogeneous. It is not clear to us why this happens. One can hypothesize that when the agents can only achieve a limited number of tasks then there are fewer agents that can do each task. That is, if an agent that can do a task is willing to do it even if it means its team will get lower utility then the system as a whole will be able to move to another new state. Otherwise, if the agents are not willing to take a task then the system is much more likely to get stuck early in the search. Similarly, when the agents can do many tasks then we have many agents that can do any one of the tasks. As such, even if one agent is unwilling to take on some particular task there is a good chance that another agent will be willing to do so. The search would then move to another state.

A fifth feature of note is the fact that very often the $L = 0$ and $L = 1$ curves are the worst performing. This indicates that neither extreme is ever the best choice for a system. That is, agents should never be completely selfish or completely selfless. Some degree of local responsibility and cooperation are needed in order to arrive at a good system-wide solution.

Finally, in order to make sure that these results remain the same as we increase the number of agents we carried out the same experiments but using 32 agents and 64 tasks. Figure 5 shows how similar those results are to the one we just found. Of course, the time needed to carry out these experiments was much longer so we were able to do only a few cases. These and other experiments we have performed make it



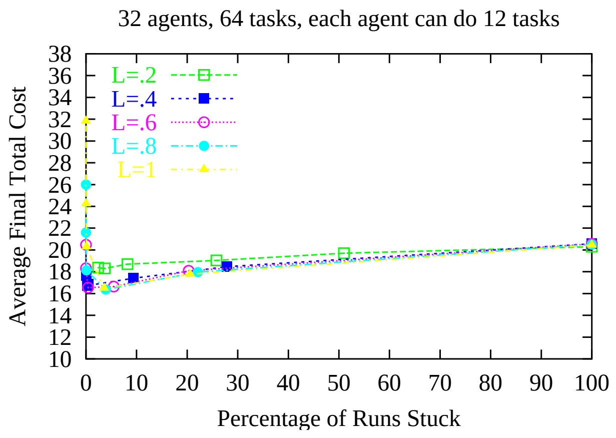


Figure 5: Results for 32 agents and 64 tasks.

clear that the insights we have gained from our initial experiments are not an artifact of that particular setting but are indeed a general phenomena which we expect to see for a wide variety of agent and task instantiations.

3.4 Random Landscape Individual Cost Functions Results

We repeated the tests from the previous sections but this time giving each agent its own cost function. This change has the effect of drastically increasing the size of the search space the agents must search. Whereas before the only aspect of a state that figured in the calculation of the total utility was the division of tasks into subsets, now we must also consider the assignment of these subsets to particular agents. The results from these experiments are shown in Figure 6.

One difference that we notice from the previous experiments is that the overall minimum is lower. This is a striking result since the search space for this case is much bigger. However, we believe that even thought the search space is bigger, the search is helped by the fact that each agent has its own cost function means that the effective branching factor is larger. By this we mean that while the number of states that can be reached from any one state remains the same (i.e., the branching factor) the number of different global utilities that those states represent is much larger (i.e., the effective branching factor). This means that the search algorithm has more options with different cost from each state that it is in. We believe that with more options available the algorithm is more likely to find a lower cost state.

On the other hand, most of the features from the experiments with a shared cost function remain unchanged. In both cases the best solution seems to be found when the systems' dynamics are at the edge of chaos; the benefits from teaming are apparent in most of the cases; and the $L = 0$ and $L = 1$ graphs are often the worst performing. As such, we must conclude that giving each agent its own randomly-generated cost function does not have a significant impact on the features we discussed and, therefore, the lessons we learned still apply for this case.

4. CONCLUSION

We have studied the benefits of teaming and selflessness for agents engaged in multiagent search in task allocation problem spaces with randomly generated cost functions. The experimental results showed several interesting results. We found that the best solution is usually attained with a maximum loss of between .4 and .6, that is, when the agents act somewhat selflessly. These parameter values allow the search to make more exploratory moves. These values seem to have similar effects to the temperature parameter in simulated annealing. We found that an even better predictor of the effectiveness of the multiagent search is the dynamics of the agents' behavior. Specifically, we found that the best global solution is always found when the systems' dynamics lie between the ordered and chaotic regimes. That is, as we vary the value of the parameters that represent the maximum loss, the number of teams, and the number of tasks that agents can do, the systems dynamics vary from ordered, where most of the runs quickly converge to some state, to chaotic, where none of the runs seems to ever converge. The best solutions were found for those cases where only a small percentage of the runs converge. We also found that small teams generally lead to better solutions and that teaming, in general, improves the quality of the result. Finally, we showed that neither complete selfishness nor selflessness are the best solution in almost all cases.

These results are important for the design of multiagent systems. Specifically, our results on the dynamics of multiagent systems seem to suggest that further research into multiagent coordination protocols should not concentrate on protocols that lead to a "clean" fixed solution but should instead study open protocols whose interactions might never end. Open-ended interaction protocols seem more likely to enable the system arrive at a better global solution. Of course, the computational and communications cost might make this a sub-optimal solutions. The final tradeoff would seem to be domain dependent.

Finally, we also hope that this article will establish a foundation for the careful mathematical study of multiagent search problems. As we have shown, these problems often appear

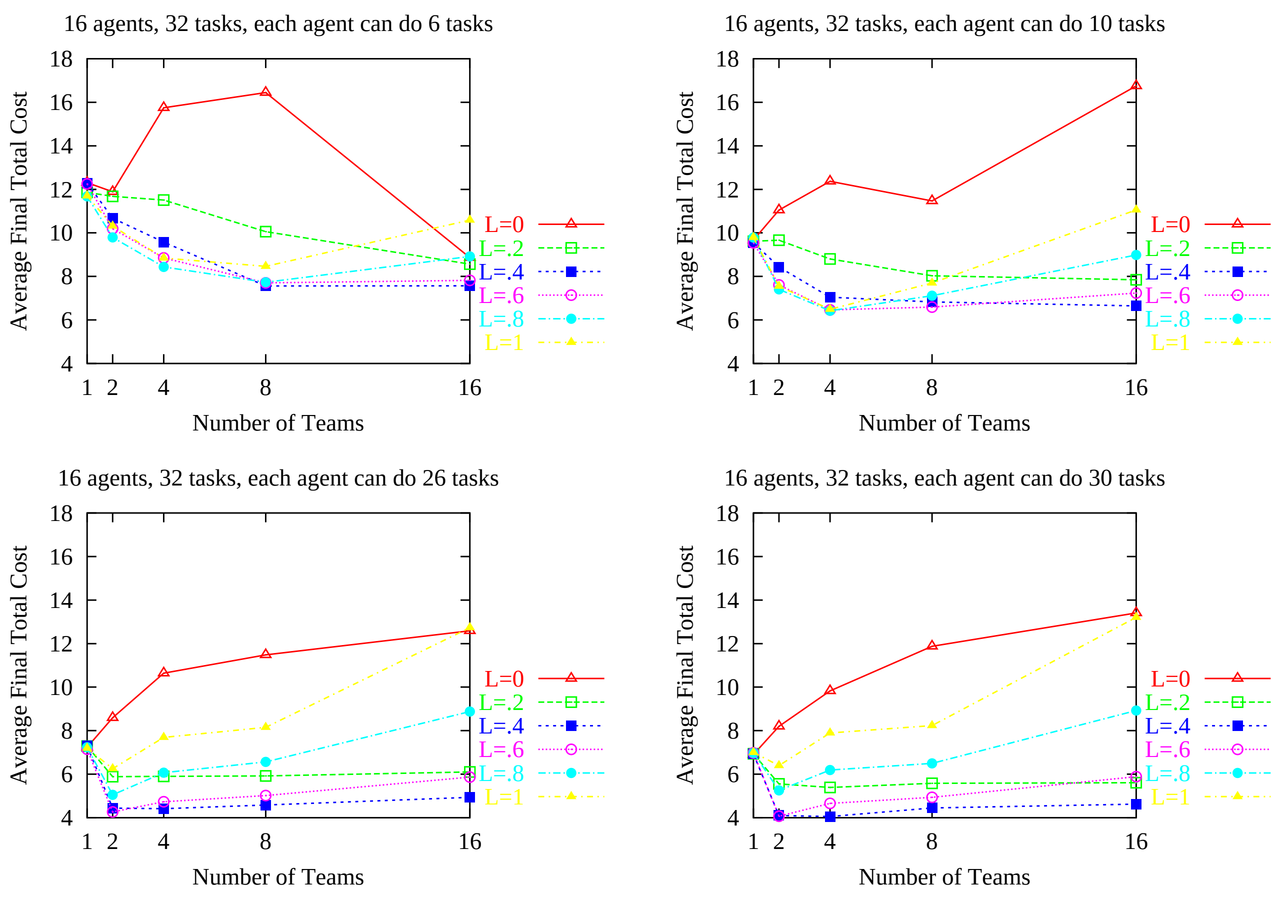


Figure 6: Results for 16 agents and 32 tasks when each agent uses its own cost function.

under different names such as task allocation, constraint satisfaction, and coalition formation. Our framework provides a rigorous platform for the comparison of multiagent coordination techniques in all these domains. We believe there is much to be gained by leveraging results from these disparate areas, as we have done in this paper.

5. REFERENCES

[1] S. H. Clearwater, B. A. Huberman, and T. Hogg. Cooperative solution of constraint satisfaction problems. *Science*, 254:1181–1183, 22 1991.

[2] L. Conway, V. R. Lesser, and D. G. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, 1983.

[3] S. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Pres, 1993.

[4] S. Kauffman, W. G. Macready, and E. Dickinson. Divide to coordinate: Coevolutionary problem solving. This reasearch has been alluded to in many subsequent publications, but I don't know if this paper has been published., 1994.

[5] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, MA, 1994.

[6] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Worst-case-optimal anytime coalition structure generation. In *Proceedings of AAAI-98*, pages 43–56, Menlo Park, CA, July 1998. AAAI Press.

[7] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, May 1998.

[8] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1981.

[9] M. P. Wellman. Market-oriented programming: Some early lessons. In S. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[10] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.