Cooperative Problem Solving

Scott H. Clearwater, Tad Hogg and Bernardo A. Huberman

Computation: The Micro and the Macro View, pp. 33–70, World Scientific 1992

Abstract

We present a quantitative assessment of the value of cooperation for solving constraint satisfaction problems through a series of experiments, as well as a general theory of cooperative problem solving. These experiments, using both hierarchical and non-hierarchical cooperation, clearly exhibit a universal improvement in performance that results from cooperation. We also show both theoretically and experimentally the super-linear speed-up that results from having a diverse collection of skills among the cooperating agents. Our results suggest an alternative methodology to existing techniques for solving constraint satisfaction problems in computer science and distributed artificial intelligence.

1 Introduction

It is widely believed that a group of cooperating agents engaged in problem solving can solve a task faster than either a single agent or the same group of agents working in isolation from each other. As a matter of fact, that cooperation leads to improvements in the performance of a group of individuals underlies the founding of the firm, the existence of scientific and professional communities, and the establishing of committees charged with solving particular problems. In the realm of computation, the emergence of massively parallel machines underscores the assumed power of concurrency for solving very complex tasks that can be decomposed into smaller pieces, and a large effort is being devoted to the design of parallel algorithms for the solution of computationally hard problems. This includes work by several authors who have pointed out the beneficial effects of cooperation on hard problems by constructing models in which a few agents communicate to accomplish a task [10, 22, 16, 5].

Nevertheless, little is known about the quantitative improvements that result from cooperation. While a large body of knowledge has accumulated over the workings of organizations, the evolution of cooperation in the biological world [3, 27] and the design of parallel computer programs [8], little is known quantitatively about what ensues from having a collection of agents working cooperatively on problem solving in general situations.

It is the purpose of this paper to elucidate under which conditions a collection of agents should best be utilized in order to solve a computational problem fastest. In order to do so we present a number of experimental results on cooperative problem solving that both test theoretical predictions and provide a quantitative assessment of the value of cooperation in problem solving [6]. These experiments were carried out by having a number of computational agents solve a set of cryptarithmetic problems and measuring their individual and global performance. These results provide a striking example of the improvements in performance from cooperation and suggest an alternative methodology to existing techniques for solving constraint satisfaction problems in computer science and distributed artificial intelligence [10]. In these experiments we compare the time to first solution for cooperative and non-cooperative agents. We also compare the scaling of speed with the number of diverse strategies used by the agents

Many problem solving tasks can be viewed as searches in large problem spaces. For realistic problems, where no algorithmic solution is known, heuristic methods are used to prune the search. Recently, a theory that elucidates the performance of cooperative processes searching through a large problem space was developed [14]. It showed that cooperative searches, when sufficiently large, can display universal characteristics, independent of the detailed nature of either the individual processes or the particular problem being tackled. This universality manifests itself in two separate ways. First, the existence of a sharp transition from exponential to polynomial time required to find the solution as heuristic effectiveness is improved [13]. Second,

the appearance of a lognormal distribution in individual agent's problem solving effectiveness. The enhanced tail of this distribution guarantees the existence of some agents with superior performance. This can bring about a combinatorial implosion [17]. In cases where the diversity of problem solving skill increases as the number of agents increases, a super-linear speed-up is observed

Before discussing the results of this paper it is important to understand what is meant by cooperation in problem solving. Cooperation involves a collection of agents that interact by communicating information to each other while solving a problem. The method of communication can be of any form, e.g., it may be through broadcasts or through access to a centralized source of information. The agents may be loosely aggregated as in a committee, or may be more formally organized as in a hierarchy. The information exchanged between the agents may be incorrect, and should sometimes alter the behavior of the agents receiving it. An example of cooperative problem solving is the use of a genetic algorithm [12] to solve a problem. In a genetic algorithm members of a population exchange pieces of themselves or mutate to create a new population to improve the overall performance of the entire population. Another example is a neural network where the outputs of connected neurons affect the output of the neuron receiving the outputs. An expert system using a knowledge base of facts with a number of processes accessing and *modifying* the knowledge base is also an example of cooperative problem solving. If no modification to the knowledge base takes place then the processes do not interact and there is no cooperation.

The next section introduces the search problem used in our experiments as well as some underlying assumptions about the class of problems we are concerned with. §3 discusses various kinds of search methods. §4 gives the experimental results and §5 gives theoretical derivations for the various kinds of behaviors expected. The paper concludes with a section where we discuss the implications of this work.

2 Constraint Satisfaction Problems

Constraint satisfaction problems lie at the heart of human and computer problem solving [20, 21, 15]. These are problems in which values must be assigned to a set of variables such that a number of conditions (the constraints) are satisfied in order for the assignment to be a solution. In most cases of interest, no direct solution method is known and one must resort to searching through a large number of possible assignments to find those that satisfy all the constraints. A *state* in the search is a set of assignments for all the variables and a *partial state* has only some of variables assigned. These search problems can be characterized by the total number of states in the search space, T, and the number of solutions, S. Let N be the number of agents or processes involved in the search. In a fully parallel system, each of these can correspond to a separate processor, but they could be simulated by a smaller number of processors as well. To be interesting the problems should be intrinsically difficult so that a single agent can't be expected to solve them in a few steps. This requires that there are many states and that solutions be relatively rare, i.e., $T \gg S$. Moreover, to have any motivation for considering cooperative problem solving, there can't be so many available agents that they could quickly examine all search states independently, i.e., we should have $T \gg N$. Since for typical constraint problems the number of states T grows exponentially with the size of the problem (e.g., the number of variables), these requirements are often satisfied in practice.

As a concrete constraint satisfaction problem for our experiments, we used the familiar problem of solving cryptarithmetic codes. These problems require finding a unique digit assignments to each of the letters of a word addition so that the numbers represented by the words add up correctly. An example is the sum: DONALD + GERALD = ROBERT, with one solution, given by $A = 4$, $B = 3$, $D = 5$, $E = 9$, $G = 1$, $L = 8$, $N = 6$, $O = 2$, $R = 7$, $T = 0$. In general, if there are n letters and the sum uses base b arithmetic then there are b^n possible states. However, not all of these correspond to the requirement that letters represent distinct digits. The requirement of a unique digit for each letter means that there are $\binom{b}{n}$ ways to choose b values and $n!$ ways to assign them to the letters, which reduces the total number of search states to $n!\binom{b}{n} = b!/(b-n)!$. Thus the above example, which has 10 letters and uses base 10 arithmetic, has 10! states in its search space. In all our experiments we used base 10 arithmetic.

Solving a cryptarithmetic problem involves performing a search. Although clever heuristics can be used to solve the particular case of cryptarithmetic [18], our purpose is to address the general issue of cooperation in parallel search using cryptarithmetic as a simple example. Thus we focus on simple search methods, without clever heuristics that can lead to quick solutions by a single agent. This is precisely the situation

faced with more complex constraint problems where extremely effective heuristics are not available. The speed at which an agent can solve the problem depends on the initial conditions and the particular sequence of actions it chooses as it moves through a search space. This sequence relies on the knowledge, or heuristics, that an agent has about which state should be examined next. The better the agent is able to utilize the heuristics, the quicker it will be able to solve the problem. When many agents work on the same problem, this knowledge can include *hints* from other agents suggesting where solutions are likely to be. Our results show that the performance of agents collaborating in constraint satisfaction problems is highly enhanced when compared to that of the same of group of agents with no interactions.

More specifically, in our cooperative experiments agents wrote the hints they discovered to a blackboard which could subsequently be read by others. This method for agent interactions, described in §3.5, differs from a case reported earlier in which the hint blackboard was filled at the start and a search algorithm chose the hints from the blackboard [9]. In that analysis, the hint blackboard consisted of orderings of rule antecedents (hints) for use in a rule learning program employing a depth-first search strategy.

Others have also studied multiagent solutions of cryptarithmetic, such as Kornfeld [17] in his implementation of parallel heuristic search. His system employed some search heuristics that estimated whether a particular agent would return useful information in a short period of time. More resources were allocated to the agents that supplied information with the least amount of resource use. His results showed an average improvement over single agent performance (typically by a factor of three using a concurrency of four which was found to be the best empirically). He also pointed out the usefulness of diversity "to increase the likelihood of discovering assumptions that can be made that will lead to valuable information quickly". This work however, contained no mathematical analysis of the process.

Blackboard systems have been widely used in parallel problem solving as in Rice et al. [22] who used the domain of aircraft tracking by passive radar to measure the performance of parallel blackboard systems using rule-based expert systems. In Poligon, their simulations included detailed implementations of control and communications. The Poligon blackboard itself was also distributed. A rapid increase in the speed with the number of processors was also found, but with an eventual leveling off due to various overhead costs involved with communications and limits of serialization.

3 Search

The behavior of a collection of agents engaged in search to solve a problem is determined by a number of characteristics. These include the overall performance measure for the group, the search methods used by the individual agents and whether individual efforts are combined (and if so how). In this section, we present several alternatives for these characteristics, and illustrate them in the context of our cryptarithmetic example. In particular, these include the ones used in our experiments and show how many kinds of complex search methods can be studied in a simplified fashion with cryptarithmetic. In many cases we found that the qualitative behavior is not strongly dependent on the specific individual search strategy used.

An agent operates in a sequence of steps in which it picks a new state (possibly incorporating hints) and tests whether it is a solution. If an agent finds a solution the first time it examines a state, it solves the problem at step 1, and so on. Specifically, a step consists of some event (e.g., pick and new state at random and evaluate it, or select a hint and incorporate it) and how long it took to complete. Thus the general description of a multiagent search is a sequence of events (specifying what was done, e.g., a hint generated and which agent did it) as well as the time they occurred.

By viewing the search as a series of events, we get a description that applies equally well to single or multiple agent searches. In the latter case, the distinction between independent and cooperating agents is what prior events influence a given agent's behavior: an independent agent's behavior is determined only by its own prior events, while a cooperating agent's behavior depends on that of others as well. As discussed in §5.5, this also allows collections in which agents choose to be independent or to cooperate, which is relevant when there are dynamically changing costs as well as benefits to cooperating.

3.1 Individual search

There are a number of search methods an individual agent can use to solve a problem. We outline a few of them here including the ones we used in the experiments. Specifically, we focus on relatively simple methods

to allow us to investigate the improvement due to cooperation even for unsophisticated agents, as well as to mimic the situation faced in more complex search problems in which effective heuristics are not available.

There are two general kinds of individual search methods: deterministic and probabilistic. Although our experiments all involved probabilistic search it is useful to highlight the differences between the two approaches. In deterministic searches, the states are examined in some pre-specified order until a solution is found. In this case any heuristics or extra knowledge gained from hints can be viewed as directly changing the number of states remaining to be examined before the solution is found. Probabilistic searches, in which states to examine are chosen at random, are more complicated. Instead of directly changing the number of states to examine before a solution is found, extra knowledge changes the probability that a solution will be found on each subsequent step.

The most straightforward search method, used in most of our experiments, is generate and test. In this case, each step generates a complete state (i.e., assignments to all the variables in the constraint problem) and tests whether it is a solution. This generation can be done in a simple pre-specified order or new states can be generated randomly. In random generation, states can be selected completely at random or the selection can be restricted to only states that have not yet been examined. The latter case avoids some unnecessary search and guarantees the search will terminate after all search states are examined, but does introduce an additional requirement of storing previously examined states and the cost of checking that they are not subsequently generated.

Other restrictions on the generation of new states are possible as well. For instance, the assignments to all the letters can be replaced in one step (which we refer to as "jumping" around the search space) or some assignments can remain unchanged, with the extreme case being a change to only a single assignment ("walking"). Walking rather than jumping through the space preserves the property that an agent near or far from a solution is still fairly near or far, respectively, after one step. Both the single letter-digit random assignment and the complete new assignment method were used in our studies.

Another common search method is backtracking which involves partial states. With this method, some ordering of the variables is selected (e.g., either fixed in advance or chosen randomly) and partial states are constructed using this ordering until a full solution is found or enough assignments are made to violate one of the constraints indicating that there is no solution corresponding to this partial state. This method is useful for many constraint problems, where these constraint violations occur well before all assignments have been made and thus backtracking avoids a considerable amount of unnecessary search. In our experiments, we use this depth-first backtracking search to compare the benefit of cooperation on easy and hard search problems.

These basic methods can be improved with the use of heuristics to guide the selection of states. An important class of heuristics uses information obtained in prior steps of the search to suggest choices. Such a class of methods allows us to directly evaluate the effect of cooperation. In a non-cooperative search, an agent using such a method could only use information that it had previously found itself, while cooperative search allows the agent to use information found by others as well.

3.2 Search in cryptarithmetic

The basic search paradigm we have used in the cryptarithmetic problem is random generate and test with replacement. For example, when the agents jump through the search space, at each step a set of possible letter-digit assignments is generated and tested to see if any of them add correctly.

As an example of our search method consider two agents trying to solve the problem AB + AC = DE. This problem has $10!/5! = 3024$ possible states and 144 solutions (determined by exhaustive search). In the first time step, each agent selects a random set of letter-digit assignments such that no digit is assigned to more than one letter. Suppose the state, i.e., letter-digit assignments, of the first agent is $A = 4$, $B = 2$, $C = 7$, $D = 3$, $E = 9$. In this case the assignments do not correspond to a solution since 42+47 does not equal 39. However, the right most column, $\mathrm{B} + \mathrm{C} = \mathrm{E}$ $(2 + 7 = 9)$, does add up correctly so that the agent's state is *partially* (or *locally*) correct. Partial correctness includes cases where a carry has been brought over from the previous column or may be sent to the next column. Note that although a particular column may be locally correct, it may not lead to a solution. Suppose the second agent's state is $A = 3$, $B = 5$, $C = 8$, $D = 7$, $E = 0$. In this case neither column adds up correctly. We will return to this example when discussing how agents behave when in a non-cooperating or cooperating environment.

For a search method that uses previously generated information, in our cryptarithmetic studies we used hints consisting of letter-digit assignments in columns that add correctly. As described more fully in §3.5, these hints were posted to a blackboard and were then made available to one or more agents depending on the overall search method. Agents used the available hints to select their next state. In a non-cooperative search, an agent using this method could only use hints that it had previously found so that each agent had a separate blackboard containing only the hints it had found. Cooperative search allowed the agent to use hints found by others as well. When each agent could communicate equally with the others (a "committee" organization) a single central blackboard contained all the hints and was available to all the agents.

For an example of the effect of hints, in the case discussed above, the first agent had one column correct (3 letters: B, C and E). If these letter assignments do lead to a solution, or are treated that way by the agent, then there are only two letters that need to be assigned from 7 possible choices. Thus the agent went from a search space of size 30240 to one of $7!/5! = 42$ states, a reduction by a factor of nearly 1000.

3.3 Relating search steps to time

So far we have presented the individual searches as a series of steps. To relate this to actual time to complete the search, we must specify the time required to perform the computation necessary for each step. For multiple agent searches, we must also relate the step times of different agents.

The simplest case is to assume that the computational work required for each search step is the same and that all agents are synchronized. This is especially appropriate for simple search methods which don't require evaluating complex heuristics and when all agents share a single processor. This situation is described by uniform synchronous time steps for all the agents.

Both of these conditions can fail to hold in more realistic cases. In a distributed environment, there may be communication delays and slightly different clock speeds so that the agents are not rigidly synchronized. On the other hand, the individual search steps themselves may take different times, e.g., some search decisions may be fairly simple to evaluate while others require extensive computation. Or processing hints from other agents could require more computation than simply proceeding with the next step of an individual search method. The most complex situation arises when the problem to be addressed (e.g., robot navigation) involves constraints that change unpredictably and asynchronously due to external events (e.g., changes in the physical world). In such cases, the agents operate asynchronously and the time required for each step can vary.

While a detailed model of these different effects is possible, for simplicity in our experiments we model both asynchronous agents and variation in the time to complete individual steps by a simple Poisson process in which steps are completed at a given average rate λ (which we take to be one in all our experiments). In practice, for the regime in which we are interested (i.e., the problem is sufficiently hard that agents, even if they cooperate, are very unlikely to find a solution in just a few steps) this Poisson process gives behavior qualitatively similar to that of uniform synchronized steps. For much of our analysis processing hints was assumed to cost the same as simply generating new states, which is reasonably valid for cases where the agents select hints randomly off the blackboard.

3.4 Non-cooperative search

In addition to individual search methods, the behavior of a group of agents depends on whether they share information. In non-cooperative search, the agents act independently. In the simplest case, each agent examines the entire search space. However, this can mean a single state is examined by more than one agent during the search. This can be avoided by partitioning the search space into disjoint parts and assigning one to each agent. In this partitioned search, agents only examine states in their assigned part of the space thus avoiding unnecessary duplicate examination of the states. In our experiments we used primarily the simplest non-cooperative case in which the search space was not partitioned and used random generate and test with replacement, as described in §3.1. Restricting each agent to examine a state at most once, as well as partitioning the search space so that a state is not examined by more than one agent, improve performance somewhat, but far less than the enhancement due to cooperation.

Continuing with the cryptarithmetic example given in §3.2 and supposing the search space is not partitioned, since neither agent has solved the problem they continue their search. This is done by selecting a

new set of letter-digit assignments to replace the previous state, even though partially correct information may be lost. This overwriting serves the purpose of preventing agents from getting stuck in locally correct states. For example, suppose the first agent randomly selects $A = 1$, $B = 3$, $C = 2$, $D = 9$, $E = 4$, which is a "jump" since all letters are given random assignments in a single step. This state has no columns that add up and represents a state that has fewer correct columns than the previous state. On the other hand, if some assignments had been left unchanged ("walking"), the partially correct information may have been preserved. The search continues until one of the agents finds a solution.

Another means of non-cooperative search is to have the individual agents save and subsequently use hints they have found themselves. In this case the agents are biased toward states which are much more likely to lead to solutions than those obtained by completely random selection.

3.5 Cooperative search

We now turn to the case of cooperation among the agents. In this case the agents exchange information regarding partial results that may be helpful to others in their subsequent search. We must specify how information is shared as well as the organizational structure, i.e., which agents communicate with each other.

In most of our work we consider a simple organization in which each agent can access the results of any other agent. In our experiments, all hints are written to a central blackboard [9] that can be accessed by all agents. There was no pre-specified limit to the size of the blackboard except that provided by the constraints of the problem. Other organizational structures are modeled by using separate blackboards to group the hints into those accessible to different subsets of the agents.

For each search step, an agent asynchronously chooses a hint randomly from the blackboard and replaces assignments in its current state with those specified by the hint. If there are no hints, or the agent has already examined the state that would result by using the hint (in the case where the agents have a memory of previously tried states), it chooses a random letter-digit assignment in the non-partitioned non-cooperating strategy described above. Once the agent obtains the new state it generates and posts all possible hints from its state, if any. Thus, assignments that work for more than one column are posted as several different hints. When random states are generated by jumping, rather than single letter replacements, there is a greater possibility of generating more hints faster but at the expense of frequently overwriting partially correct states.

In the simple example discussed previously, the procedure for cooperating agents is shown in Fig. 1. Hints for the cryptarithmetic problem were the letter-digit assignments in columns that add correctly. Suppose the first agent evaluates its state before the second agent. Since it has a column of letters that add correctly, this is posted to the blackboard as a single hint. Thus, the blackboard now contains the hint $B = 2$, $C = 7$, $E = 9$. Next, the second agent randomly selects one of the hints (in this case there is no choice because there is only one hint) and updates its current state with the hint contents, i.e., makes the letter assignments specified by the hint. To maintain unique assignments for the digits, if in the existing state a digit is also assigned to a letter appearing in the hint, then the hint assignment supersedes the existing state assignment and the letter in the state that had the duplicate digit is assigned an unused digit or interchanged with the duplicate. Thus, the second agent's initial state of $A = 3$, $B = 5$, $C = 8$, $D = 7$, $E = 0$ becomes, after utilizing the blackboard hint, $A = 3$, $B = 2$, $C = 7$, $D = 8$, $E = 9$. Thus, the second agent still has not solved the problem, and cooperation continues in this fashion until an answer is found.

Some cooperative situations involve agents with different kinds of abilities or search strategies. Such diverse communities are particularly well-suited for the use of cooperation since a particular agent may not be able to utilize all the information it generates, whereas another agent, using a different strategy, can. This exchange of information can improve performance beyond that possible without cooperation. To examine this situation we used a very simple way of introducing diversity for cryptarithmetic: specialize the agents to only examine hints of certain lengths (i.e., containing assignments to only a certain number of letters).

The effectiveness of these hints will generally depend on the search choices made by the agents. E.g., as the search progresses, agents may find better partial solutions so that hint quality increases over time; conversely, as agents get near the solution, hints become less important since they will tend to duplicate partial solutions already found.

The key difference between cooperating and non-cooperating agents is that hints effectively reduce the size

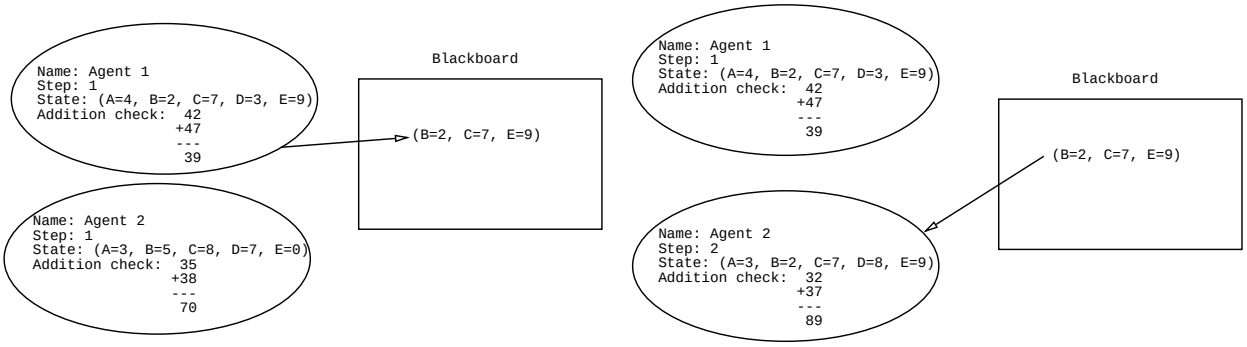

Figure 1: The simple cooperative cryptarithmetic example with two agents discussed in the text. In the left half of the figure, Agent 1 has posted a hint of length 3, i.e., one containing assignments to three letters, to the blackboard. On its next update, shown in the right half, Agent 2 uses the hint to update its state.

of the search space or increase the probability that a solution will be found on a given step, for deterministic or probabilistic searches, respectively. In the former case the cooperative search focuses the agents on much more plausible courses of action. In the latter case, the agents have an enhanced probability of finding a solution at a given step. With many agents solving a problem and starting at different locations in the search space there is a chance that some of them will find hints worth communicating to other agents, hints that can in turn be used to reduce subsequent search.

We also studied hierarchical organizations of cooperating agents to see how the benefit of cooperation depends on the organizational structure. There are several motivations for this. First, all large organizations work as hierarchies with restrictions on the communication of information. In addition, there are limits on the size of any group of collaborating agents. Among these, we mention communication costs, finding relevant hints, and the problems associated with sustaining collaboration when agents incur a cost for participating in cooperative problem solving. Communication costs become particularly important when a large organization is involved. In particular, communication costs eventually overwhelm a committee. Although computer communication bandwidths are ever increasing and do exceed human bandwidths, there is a point where communication costs become excessive for the performance desired, especially in distributed problem-solving environments. In these situations it becomes advantageous to structure the organization in the form a hierarchy. This restricts to some extent the allowed communications while still permitting a flow of information. Specialization of tasks permits more efficient use of hints between the appropriate groups of agents and gives another motivation for using a hierarchy. This speeds up problem solving by allowing agents to work on sub-problems without having to worry about solving the entire problem by themselves. Experimentally, hierarchical organizations were created by giving each agent a separate blackboard and specifying which other agents could access it based on the organizational structure.

3.6 Performance measures

Before turning to our experimental comparison of cooperating and non-cooperating agents, we must specify how the performance of a group of agents is to be measured. The appropriate performance measure depends on the nature of the problem, and our simple cryptarithmetic problem can be used to provide examples of various measures.

In many cases, one is interested in finding a single solution to the problem and each agent is individually capable of finding a complete solution. This means that the search is completed as soon as one agent finds a solution. The appropriate overall performance measure is then just the time required until some agent in the group finds a solution.

In other cases, several or all solutions to the problem are required so the group performance is determined by when all the required solutions are found. Moreover, the processing power devoted to those agents that finish early (i.e., before the required number of solutions is found) may become available for the remaining agents.

When computational resources are limited and the problem is intrinsically very difficult (e.g., the traveling

salesman problem [7]) one must often be content with obtaining only an approximate solution. In such satisficing search problems, the performance measure is the quality of the best state found by the agents in a given time rather than whether an optimal or exact solution state is found.

A more complex case arises when there are many related problems to solve. The problems can be assigned to individual agents. When an agent completes its problem, it is given another, or may be reassigned to an unrelated task and thus provide no additional help to the other agents. An example is the processing of scanned images of many pages of a single document in which each page could be assigned to an agent. Because these pages are related, information gained from processing one page could potentially be useful as a hint to other agents. An appropriate performance measure could then be how many problems (e.g., pages) the group of agents solves in a given amount of time. This is determined by the distribution of individual performance times for the agents rather than just the speed of the fastest ones.

4 Experimental Results

In this section we present our experiments on the behavior of multiagent search for cryptarithmetic. Although our simulations considered multiple agents, we could just as easily have considered a single agent consisting of many parts. In fact, all the simulations were done on a single processor running a single process. The important point is our observations of the *ratio* of the performance between a cooperative multiagent simulation and a non-cooperative one should not depend on whether we use a single processor or many. What is most important to the increase in performance is the diversity of approaches available by having many agent processes.

4.1 Problem Solving by Committee

In this section we present results from many experiments involving a committee of agents solving cryptarithmetic problems.

4.1.1 Super-linear Speed-ups and Diversity

Amdahl's law [2] predicts at most a linear speed-up in performance with the number of processors. However, this places no restriction on its software equivalent: the speed-up due to adding additional processes with diverse solution methods. One of the most dramatic effects we have seen is a super-linear speed-up due to diverse cooperating agents. While it is difficult predict what sort of diversity will lead to significant super-linearity it is possible to test the prediction itself. Fig. 2 shows a super-linear speed-up as new strategies are added to a collection of cooperating agents along with comparisons to two groups of non-cooperating agents: those that use the same strategies of interpreting the hints and those that use random generate and test with no hints.

To illustrate the effect of diverse strategies on problem solving we chose to use the length of the hint for types of diverse strategies, although others could have been used. Three strategies were chosen, corresponding to a non-overlapping partition of hint lengths. In particular, the first strategy used only hints that had length equal to 5. The second strategy used hints of length 4. For a collection of agents using the two strategies, half use the length 5 strategy and half use the length 4 strategy. The third strategy used hints of length two or three. In a collection of agents using all three strategies, each strategy would be used by one-third of the agents. The super-linearity comes from the fact that agents with a given strategy can contribute hints that it cannot use to agents that can utilize the hints by using a different strategy. This leads to a non-linear increase of the number of hints available. The cooperating agents and the non-cooperating agents with no blackboard or memory of previous states define a performance envelope that corresponds to the extreme cases of performance that may be expected from a collection of agents and strategies. To summarize, for cooperative diverse agents the diversity of problem solving strategies leads to a super-linear speed-up in problem solving.

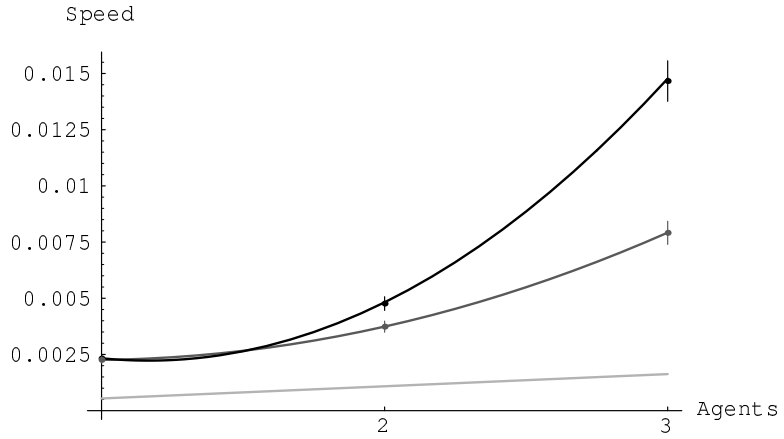

Figure 2: Adding diversity leads to a super-linear speed-up in solution time. Average speed of the first finisher, defined as the inverse of the average time required until some agent first finds a solution, for 200 runs as a function of the number of agents employed trying to solve WOW + HOT = TEA. Steps were related to time by a Poisson process with average rate $\lambda = 1$. The bottom linear line is for the case of non-cooperating agents, each using the same random generate and test strategy with no memory of hints. The two non-linear curves correspond to the case where each agent uses a different strategy for utilizing hints. The middle curve corresponds to the case where the agents were non-cooperating, had their own blackboard and did not revisit states. The dark super-linear curve fits is the cooperating case where the agents shared one blackboard and did not revisit states. Each error bar corresponds to the statistical error of the mean of the measurement.

4.1.2 The Distribution of First Finishing Times

The previous experiment showed the behavior of the average time to first solution. Due to the probabilistic nature of the search, and the exact choice of hints, the time required to find a solution will vary substantially from run to run. Thus while we see that cooperation improves average performance, it is also of interest to examine the distribution of finishing times. A comparison of the distribution of fastest finishers for cooperating agents and non-cooperating agents is shown in Fig. 3. It shows that for cooperating agents the probability for finishing sooner is higher so that the benefit is seen for most runs, not only on average.

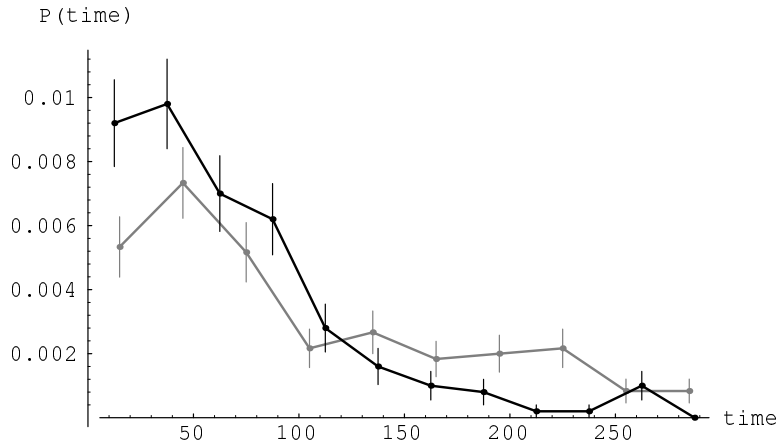

Figure 3: The distribution of first finishing times is peaked towards faster times for the cooperating case. The data used are the same as in Fig. 2 for the three agent case. The dark points connected by black lines correspond to cooperating agents and the gray corresponds to the non-cooperating case.

4.1.3 The Distribution of All Finishing Times

The previous discussion is relevant to the case in which the time to first solution is of most interest. In other kinds of problems, all the agents participate in a series of tasks and the amount of work the entire group can complete in a given amount of time is the relevant performance measure. This leads us to consider the distribution of solution times. Moreover, examining this distribution provides a connection with the theory of §5.

In this set of experiments all the agents used the same strategy but examined different states due to their different initial locations in the state space and their choices from the available hints. Our experiments explored several possible cases of agent interaction. Specifically, the speed distribution for the cases a) non-cooperating agents, b) non-cooperating agents with a partitioned search space, and c) cooperating agents, was compiled for agents trying to solve the problem WOW + HOT = TEA. Fig. 4A shows the resulting speed distribution for a typical run for each of the three cases, where speed is defined as the inverse time to solve the problem for an agent. The striking overall improvement in performance becomes apparent once we note that the speed is up to 100 times larger for the performance distribution of the cooperating agents than that obtained for the non-cooperating case and about 10 times larger than the partitioned case. However, this speed improvement is also partly due to a better search method used by the cooperating agents. The distribution for the cooperative case using the results from ten batches is shown with a linear scale in Fig. 4B. Note that the fastest performers, the average performers and even the entire distribution for the cooperating agents is shifted towards higher performance. We should point out that the actual speed-up for the cooperative case, when the problem is easy or the hints are very good, can vary greatly from run to run because it is very sensitive to the quality of the first few hints posted to the blackboard. Similar results were obtained on other cryptarithmetic problems that used between 5 and 10 different letters and had from 1 to over 100 solutions.

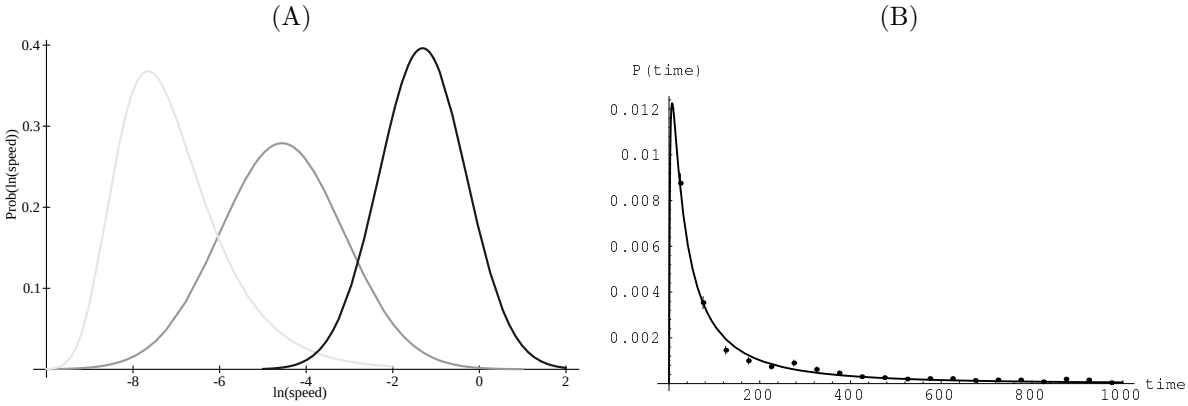

Figure 4: (**A**) Probability density distribution of ln(speed) obtained by fitting typical runs for agents trying to solve WOW+HOT = TEA. The left most curve is for 100 non-cooperating agents, the center curve is for partitioned search (using 720 partitions and agents) and the right most curve is for 100 cooperating agents. The left curve corresponds to a exponential time distribution and the other two to lognormals (The connection between the lognormal and the finishing time distribution is given in §5.). The conversion to a log scale allows all three cases to be shown on the same plot and also makes them appear similar to a normal distribution. (**B**) Probability vs. finishing time for 100 cooperating agents and 10 runs solving WOW + HOT = TEA. Each agent chose hints from the blackboard using one of five possible strategies. Each strategy selects hints of only a single length ranging from 3 to 7. The error bars are statistical. The curve is a lognormal distribution using the data to obtain the parameters. Although the fit is not very good, based on a significance test, it certainly has qualitatively the behavior expected from the discussion of §5 which assumed the blackboard has had time to warm up before agents start solving the problem.

We also consider the effect of diversity on the distribution of all finishers. The speed was fastest when all the agents had access to any hint. Although this corresponded to lower diversity (i.e., all the agents used the same strategy), it was compensated by a much higher average performance. Further, it was observed that by increasing the diversity of the agents' hint interpretation, the mode and median of the speed distribution were reduced. This shows that with greater diversity the higher performers benefit while the lower performers

suffer.

4.1.4 The Effectiveness of Hints

It also worthwhile to note the effect of cooperation as the problems become more difficult. One way of measuring the difficulty of problems is by the complexity ratio, T/S. The table shows the relative speed for the first finisher of 100 agents for four problems of vastly different complexities, i.e., the ratio of speed of the cooperative to non-cooperative cases. The data for the cooperative case came from simulation runs while the behavior of the non-cooperative case was obtained theoretically (from Eq. (2) of §5). Note that as the problem becomes more difficult the importance of cooperation in speed-up is increased. The relative increase becomes even more startling when one considers that the fraction of hints posted on the blackboard that are subsets of *any* of the solutions (there may be more than one solution) decreases as the problems become more complex. Thus the high performance is due to some agents finding combinations of hints that lead to solutions.

problem	ratio of speeds	T/S	fraction of hints that are subsets of solutions
AB + AC = DE	7	210	0.9–1.0
WOW + HOT = TEA	45	1844	0.5–0.6
CLEAR + WATER = SCOTT	145	181440	0.1–0.2
DONALD + GERALD = ROBERT	315	3628880	0.004

Another way of studying the effect of cooperation vs. problem complexity is to vary the effectiveness of the search performed by the agent by itself, without utilizing the hints from the other agents, or the *self-work*. For example, suppose that when the agents are not using hints they perform a depth-first backtrack search, each using a randomly selected ordering for the variables. During the depth-first search the agents have the opportunity to prune partial states which do not lead to any solution. For example, if some columns do not add up correctly there is no point in considering assignments to uninstantiated letters for this state. Whenever a hint comes along it overwrites the current partial state, in the same manner as for the agents using simple generate and test, so that there may be very large jumps through the search space. We can simulate the effect of this pruning by probabilistically pruning partially assigned states that are known not to lead to a solution. (We can do this with cryptarithmetic because we can generate all the solutions.) When the probability of pruning is small this corresponds to difficult problems because the agents must instantiate nearly all the letters before pruning. The results of this study, which are shown in Fig. 5, show the greater relative importance of cooperation for harder problems.

4.1.5 Prior Knowledge

In many practical applications the agents start with some prior problem-specific knowledge. We now consider the effect of the quality of the agents' initial knowledge of the search space. This was modelled by the inclusion of hints on the initial blackboard. The effect of a non-empty starting blackboard is to significantly increase the number of unique solutions found. This is because random hints will likely point to different solutions, whereas an empty blackboard implies that the attention of the agents is highly focused by the first few arriving hints. Also, a non-empty starting blackboard can lead to a smaller diversity because with many hints already available, the importance of hint selection strategy becomes less important.

The initial contents of the blackboard also affect the overall performance of the system. If hints are misleading, the agents will be slowed; if, on the other hand, hints are useful then the overall performance will be improved, especially for the faster agents. In general, as the number of hints present on the initial blackboard was increased, the statistical fluctuations in the effectiveness of the hints decreased and ensuing performance was enhanced.

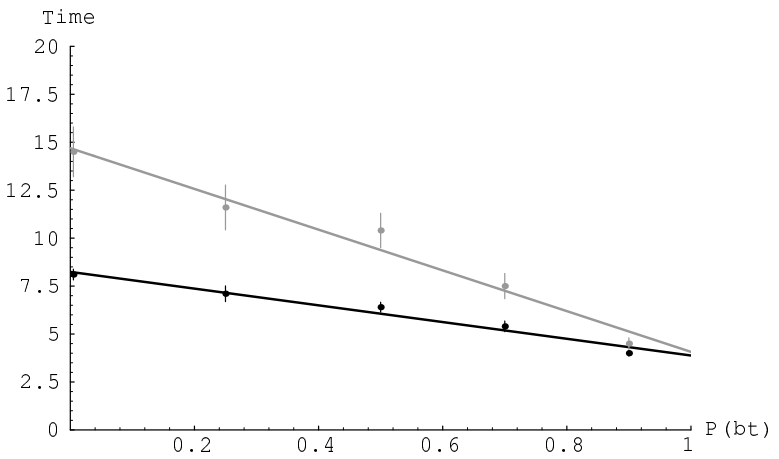

Figure 5: Cooperation works best for harder problems. Time to first solution as a function of decreasing problem hardness. Specifically, the plot shows the average time to first solution for 100 agents solving AB + AC = DE as a function of the probability of pruning, $P(bt)$, a state that is known not to lead to a solution. The left side of the plot corresponds to "hard" problems where pruning of the search space is very poor, and the right side of the plot corresponds to "easy" problems where pruning is very effective. The light line is for the case of non-cooperating agents, in this case a depth-first search. The dark line is for the case where the agents spend 80% of their time doing depth-first self-work and 20% cooperating, i.e., using hints from the blackboard. The lines show the best linear fits to the data. The data points correspond to the average solution time from 50–100 runs. The error bars are the error of the mean.

4.1.6 Fluctuations

We have shown that cooperation can lead to a large performance increase, but how consistent are these predictions? By running the same problem numerous times we can get an idea of the fluctuations that can be expected. With an initially empty blackboard the impact of the quality of the early hints can be easily observed. Fig. 6 shows two separate runs of 100 agents solving the problem CLEAR + WATER = SCOTT. Although the two runs agree with a lognormal distribution, they are quite different in terms of the speed at which they solved the problem. These fluctuations are amplified by the multiplicative nature of cooperation. We should point out however, that these differences became smaller as the problem became more complex and the number of agents increased.

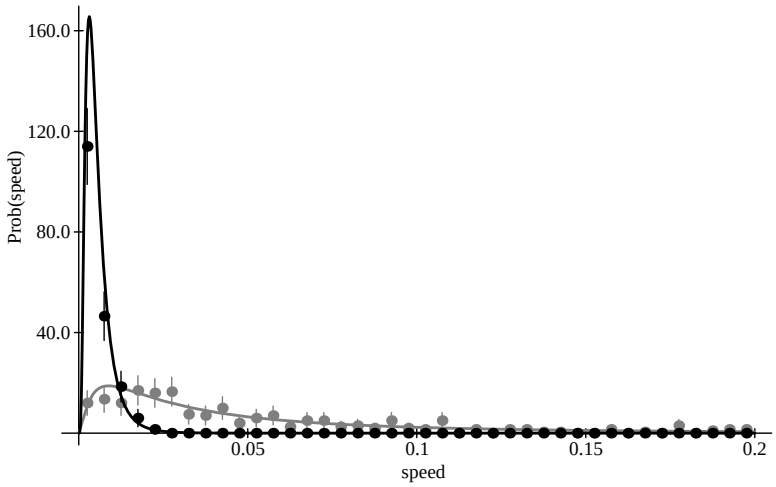

Figure 6: The quality of hints can lead to vastly different lognormal distribution parameters. The data are speed distributions from two separate runs of 100 agents solving the problem CLEAR + WATER = SCOTT. Note that both distributions fit well to a lognormal. The error bars are statistical.

It is possible to study the effect on the overall solution time distribution of making solutions available to the agents on the blackboard. The option of not posting complete solutions had the effect of slowing down the slower finishers. When the solutions were posted for the problems WOW + HOT = TEA and CLEAR + WATER = SCOTT, about half the agents found the solution by reading one directly.

4.1.7 The Cost of Accessing Hints

So far we have considered the access to hints for the agents to be cost free. In real situations, this is not the case and it useful to see the effect that access cost has on the speed distribution. When access cost was introduced in our experiments (measured in extra time per step) the functional behavior still followed a lognormal distribution, albeit peaked at slower speeds than the one without access cost. Fig. 7 shows the effect of accessing hints where the time cost of an access is proportional to the length of the hint accessed. Notice that, although the time cost significantly slows down time to solution, cooperating agents can still solve the problem much faster than non-cooperating ones.

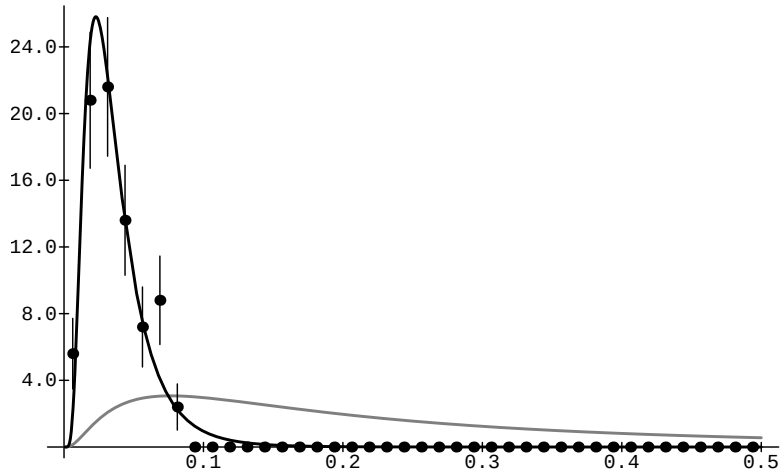

Figure 7: Adding a cost for accessing hints slows down problem solving. The plot is the probability of speed versus speed. The data and fitted curve are for a typical run of 100 agents solving the problem WOW + HOT = TEA. Each hint access causes a time delay in time units equal to the number of letters in the hint. The average evaluation between blackboard hint accesses was one time unit. The gray curve shows the fitted curve from another typical run but with no cost for hint access for comparison.

4.1.8 Satisficing Searches

In many situations it is not possible to wait until a full solution is found because of a time constraint. In these cases the "best" partial solution found at the stopping time is called at "satisficing" solution [25]. One way to study satisficing searches within the cooperative framework described above is by examining the best partial solution a collection of cooperating agents have versus a set of non-cooperating ones as a function of time. Fig. 8 gives a comparison of satisficing searches for cooperating versus non-cooperating agents.

4.1.9 Other Observations

We now make some general observations concerning our implementation of cooperative problem solving. A notable difference between non-cooperating agents and cooperating ones is in the average change of the number of correctly added columns after each new hint was applied. On average, although each hint helped slightly, their overall effect was to increase the rate at which correct columns were found, leading to a huge increase in the global performance of the system. Although the size of the blackboard increased as all the agents solved the problem, there is a point at which the addition of new hints does not add useful information but merely reinforces it. This is because once a representative sample of hints has been found, the blackboard

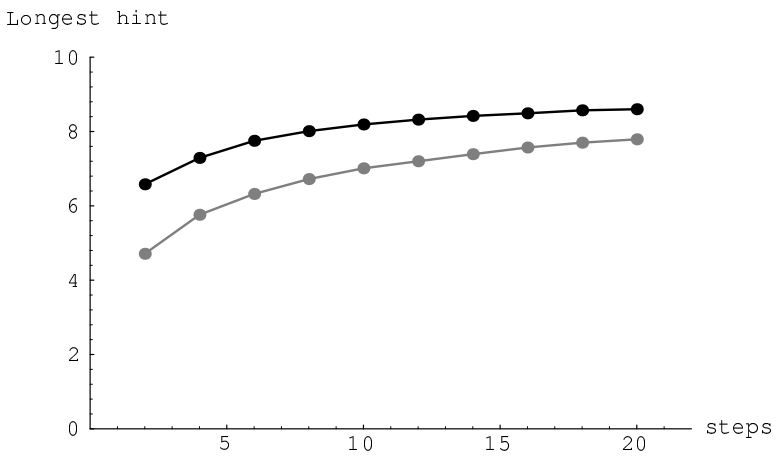

Figure 8: Cooperation improves the results of satisficing search. Comparison of a satisficing search for a given number of steps for 10 agents solving the problem DONALD + GERALD = ROBERT. Each cooperating agent was able to access any of the hints on the global blackboard. Each non-cooperating agent was able to access only the hints on its own blackboard. The vertical axis is the average, taken over 100 runs, of the maximum number of letters (out of a possible 10) that add correctly. In the cooperating case this is simply the longest hint on the blackboard at a given time. The cooperating agents (black dots connected by black lines) are seen to maintain a higher average than the non-cooperating agents (gray dots connected by gray lines), again showing the benefits of cooperation. The errors of the mean are too small to be seen on the plot. In the non-cooperating case this is the maximum of the longest hint of each agent's blackboard.

is effectively saturated and any other hints only serve to statistically improve the hint distribution rather than adding any systematically new information. For example, consider a ten letter cryptarithmetic problem. Once an eight letter hint has been found that leads to a solution, it is not particularly useful to add another one of the same length that is redundant. In situations where the quality of the hints improves noticeably over time it will still be beneficial to add new hints since the density of good hints on the blackboard will increase, thereby improving the problem solving ability of the remaining agents. The effect of a saturated blackboard is that the size of the blackboard need not be unbounded in order to obtain significant speed-up.

4.2 Problem Solving by Hierarchical Organizations

To model the effect of more complex organizations, we considered several alternatives to committees in both the hierarchical structure and the interactions among agents. These alternatives were based on the branching factor between levels, the number of levels, cost of access between and within levels, and other parameters of hierarchies. Under certain conditions a hierarchy can perform poorly. However its performance can be improved by the addition of informal links between nodes [19]. To investigate its applicability to search, we compare behavior in hierarchies, illustrated in Fig. 9, with and without such informal links.

For search in the hierarchy we associate a blackboard with each agent, but instead of restricting agents to use only their own blackboard (a non-cooperative search), they now can post and read hints from other ones as well. Unlike the committee case involving a single central blackboard, the agents now need to decide where to send and from where to receive their hints. Thus, depending on the particular hierarchy involved different blackboards (agent's individual postings of good columns for the cryptarithmetic problem) were accessible via different costs.

In our experiments the cost of communication increased the greater the distance between the nodes in question, as is the case in real hierarchies. It should be noted that only read access was attributed a cost. Also, although write cost was considered to be free, the likelihood of posting a hint on a different level blackboard decreased the further away the two blackboards were. This situation only applied for the cases where the better hints were not restricted to higher levels.

Fig. 10 shows speed distributions for a purely formal hierarchy and a hierarchy with many informal links. The difference in speed-up between the curves depends on the costs and probability of accessing other nodes

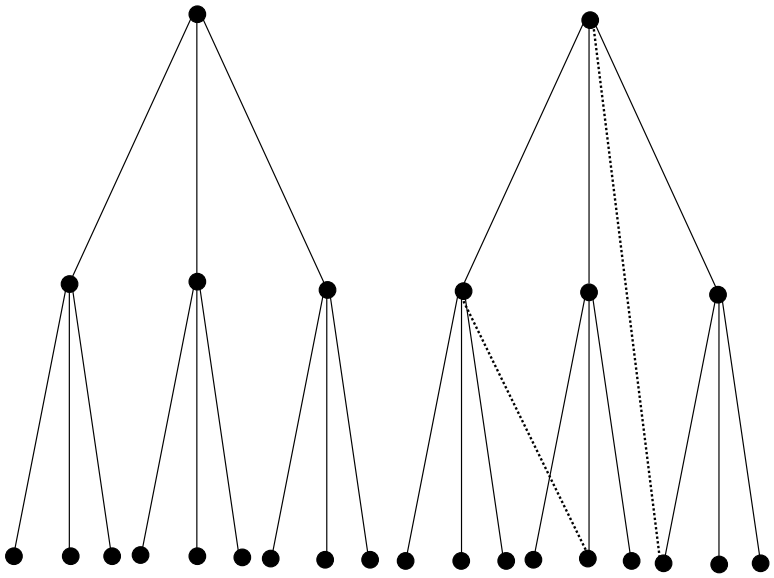

Figure 9: *Left*: A strict hierarchy with branching factor three and depth three. *Right*: The same tree with two additional "informal" links in dashes. The informal links have little or no communication cost and do not exist in a formal hierarchy.

as well as the quality of hints at a given level. The speed to solution was found to be greater for organizations that had informal links, i.e., links that violated the strict hierarchy and had zero communication cost. It remains an open question exactly how changes in organizational structure affect the benefit of cooperation.

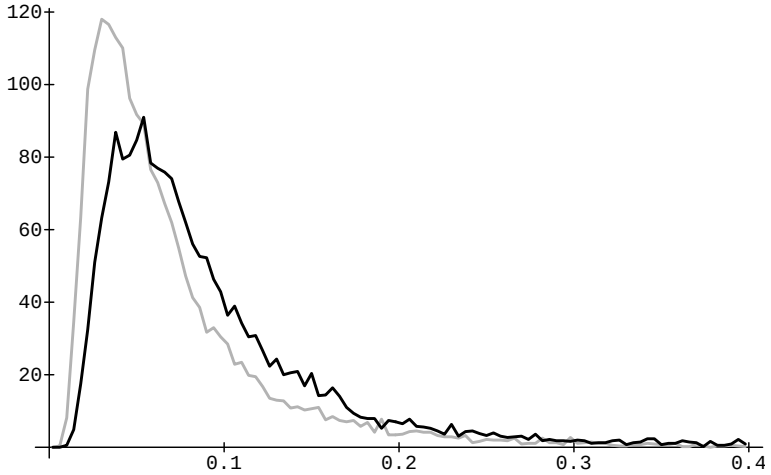

Figure 10: Speed probability distribution for 100 runs of 111 hierarchical agents trying to solve WOW+HOT = TEA. The branching ratio was 10 and depth of the organization 3. The light curve shows the case where there were sibling links but no informal links. The dark curve shows the case where the number of informal links added to each node was equal to the number of its direct descendents, siblings, and all ancestors (except the top node received no informal links).

We also found that hierarchies in which higher quality information was passed only to higher levels achieved a higher performance than those in which information could be posted anywhere. It is interesting to note that with informal links and where no communication among siblings was allowed, the top level agent solved the problem much more frequently than would be expected by its representation in the population. In cases where there was sibling communication or informal links were allowed, the agents at the lowest level solved the problem more frequently than would be expected based simply on their relative occurrence in the

organization.

5 Theory

In this section we present simple theoretical descriptions of the behavior of both independent and cooperative agents that illustrates the large benefit of cooperation. Our theory of cooperation addresses the issue of when and what type of cooperation is likely to be beneficial under certain conditions. Specifically, for a group of agents we characterize their overall performance in terms of their individual search methods and how they combine their efforts. There are a number of possibilities for each of these specifications, as discussed in §3.

We should note that the experiments contain elements that make them correspond to realistic search implementations, but which complicate theoretical description. Also, not every possible variation of the simulations can be described in a simple theoretical way. Thus, the theories presented here are not exact analogues to every simulation experiment described in the previous section, but provide a simple explanation of the general phenomenon and, most importantly, suggest it is more general than just cryptarithmetic.

5.1 Single agent behavior

The search for an individual agent consists of a series of steps in which it examines states in a search space. Its behavior can be described in terms of the probability of finding a solution at step k, $P_{\text{success}}(k)$, given that no solution was found in previous steps. In general, this will depend on what choices were made in the previous steps. The probability that a particular agent solves the problem at its k^{th} step is then given by

$$P_{\text{solve}}(k) = \prod_{\kappa=1}^{k-1}(1 - P_{\text{success}}(\kappa))P_{\text{success}}(k) \tag{1}$$

where k runs from 1 (finding the correct answer immediately) to infinity (never finding a solution). It is also useful to define the cumulative probability of solving the problem at or after step k: $C_{\text{solve}}(k) = \sum_{\kappa=k}^{\infty} P_{\text{solve}}(\kappa)$.

For example, in a deterministic search in which the solution is found at step K has $P_{\text{solve}}(k) = P_{\text{success}}(k) = \delta_{kK}$. In the case of random generate and test with replacement (so there is no memory of previously examined states), $P_{\text{success}}(k) = S/T$, where T is the total number of search states and S the number of solutions. This gives the familiar geometric distribution

$$P_{\text{solve}}(k) = \left(1 - \frac{S}{T}\right)^{k-1} \frac{S}{T} \tag{2}$$

When states examined are not revisited, $P_{\text{success}}(k) = S/(T - k + 1)$ and

$$P_{\text{solve}}(k) = \frac{S}{T - k + 1} \prod_{j=1}^{k-1} \left(1 - \frac{S}{T - j + 1}\right) \tag{3}$$

which asymptotically approaches Eq. (2) for $k \ll T$. Note that both distributions monotonically decrease with the number of steps.

5.2 Relating search steps to time

The above distributions describe the search in terms of steps. Giving results in terms of time required to find the solution requires specifying how long the steps take to compute. Specifically, let $P(k, t)dt$ be the probability for step k to take place in the time interval $(t, t + dt)$, which will generally depend on the choices made in the previous steps. This can be used to find the distribution for the *time* to first find a solution:

$$P_{\text{solve}}(t) = \sum_{k=1}^{\infty} P_{\text{solve}}(k)P(k, t) \tag{4}$$

and the corresponding cumulative distribution $C_{\rm solve}(t) = \int_t^\infty P_{\rm solve}(\tau)d\tau$ to solve the problem at or after time t. The average solution time is given by $t_{\rm avg} = \int_0^\infty tP_{\rm solve}(t)dt$, which can also be written as $t_{\rm avg} = \int_0^\infty C_{\rm solve}(t)dt$ provided $C_{\rm solve}(t) \ll 1/t$ as $t \to \infty$. Alternatively, one can describe the speed with which a problem is solved, defined as the inverse of the solution time. The distribution of speeds is just $P_{\rm speed}(s) = P_{\rm solve}(1/s)/s^2$.

As discussed in §3.3, the details of this characterization will be complicated when interactions between agents and communications costs are taken into account. The simplest case is to use synchronized uniform steps, in which case the description in terms of search steps applies directly as a description in terms of time. In our experiments we used a somewhat more complex model to model asynchrony and variation in time to complete each step. Specifically, we assumed that the agents perform their search steps at a rate λ and that the time between steps is independent of the previous steps. These assumptions give the familiar Poisson distribution for k events in time t that occur at an average rate λ, $p(k,t,\lambda) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$. The probability density for the time at which the k^{th} step occurs for a Poisson process is given by the Gamma distribution,

$$P(k,t) = \frac{\lambda(\lambda t)^{k-1}e^{-\lambda t}}{(k-1)!} = \lambda p(k,t,\lambda) \tag{5}$$

We should note that the assumption of a Poisson process gives essentially the same behavior as uniform steps in the regime of interest for our theory (i.e., when the problem is sufficiently difficult that many steps are required to solve it, even with cooperation).

One other issue is that the time required to complete each step could depend on the number of agents. In particular, if the processes all share the same processor, the effective rate at which steps are performed will be inversely proportional to the number of agents. For simplicity, we suppose each agent runs on its own processor so we can take λ to be independent of N. This choice, which corresponds to our experimental simulations, has no effect on the *ratio* of performance of various search methods, e.g., cooperative and non-cooperative.

As an example, consider the case where $P_{\rm success}(k) = p$ is independent of k, e.g., random generate and test with replacement which has $p = S/T$. Using Eq. (2) and (5) in Eq. (4) gives

$$\begin{aligned} P_{\rm solve}(t) &= \sum_{k=1}^{\infty} p(1-p)^{k-1}\lambda\frac{(\lambda t)^{k-1}}{(k-1)!}e^{-\lambda t} \\ &= \lambda p e^{-\lambda p t} \end{aligned} \tag{6}$$

and $C_{\rm solve}(t) = e^{-\lambda p t}$. The average time to solution is $1/\lambda p$. The corresponding speed distribution is

$$P_{\rm speed}(s) = \frac{\lambda p}{s^2}e^{-\lambda p/s} \tag{7}$$

These results show that the solution time for a single agent is governed by the density of solutions, i.e., the ratio of the number of solutions to the number of possible states in the search space, which we use as a measure of problem complexity.

5.3 Non-cooperative agents

We now turn to the behavior of a collection of N independent agents involved in a search. Let $P^{(N)}_{\rm solve}(t)dt$ be the probability that at least one of the N agents first solves the problem during $(t, t+dt)$ with $C^{(N)}_{\rm solve} = \int_t^\infty P^{(N)}_{\rm solve}(\tau)d\tau$ being the cumulative distribution. (Note that this is for probability densities: if discrete steps are involved, the density involves delta functions and it is easier to work directly with the probability to solve at a given step which is $P^{(N)}_{\rm solve}(k) = C^{(N)}_{\rm solve}(k) - C^{(N)}_{\rm solve}(k+1)$.) The average time to solution for the collection of agents (i.e., average time until one agent gets a solution) is

$$t^{(N)}_{\rm avg} = \int_0^\infty tP^{(N)}_{\rm solve}(t)dt \tag{8}$$

This can be expressed in terms of the individual agents' search methods. Specifically, let τ_i be the random variable denoting when agent i solves the problem. The probability this is greater than t is denoted

by $C^i_{\text{solve}}(t)$. Then $\tau = \min(\tau_1, \ldots, \tau_N)$ is the time for the first of a collection of N agents to solve the problem. With the independence of the agents, the probability this will be greater than t is

$$C^{(N)}_{\text{solve}}(t) = \prod_{i=1}^{N} C^i_{\text{solve}}(t) \tag{9}$$

i.e., to be greater than t, *all* agents must take longer than t to solve the problem.

Returning to the example of random generate and test with a Poisson step distribution in which all agents use the same search method we have, from Eq. (6), $P^i_{\text{solve}}(t) = \lambda p e^{-\lambda p t}$ for all i. This gives

$$P^{(N)}_{\text{solve}}(t) = N\lambda p e^{-N\lambda p t} \tag{10}$$

so that N independent agents all using this search method behave as a single agent going N times faster. In particular we have $t^{(N)}_{\text{avg}} = 1/N\lambda p$, i.e., a collection of N agents solves the problem N times as fast as a single one.

If instead the steps are synchronized and uniform, using Eq. (2) we get

$$P^{(N)}_{\text{solve}}(k) = (1-p)^{N(k-1)}\left(1-(1-p)^N\right)$$

which is a geometric distribution as a function of number of steps to find a solution. (Previously [6], we stated that the distribution was geometric as a function of time rather than as a number of steps. This is not formally true, although in the range of our interest it is asymptotically correct.) We should note that the behavior of this distribution, and in particular the average solution time, is the same as that given above for Poisson steps when the uniform steps are of length $1/\lambda$, provided $Np \ll 1$, which is the limit of our interest, since otherwise there are so many agents that they can independently solve the problem in just a few steps.

If the agents work in a partitioned space, the behavior is the same as above with the exception that each agent has a different probability to find a solution. That is, instead of the uniform value $P_{\text{success}}(k) = p = S/T$, agent i now has $P^i_{\text{success}}(k) = S_i/(T/N) \equiv p_i$ (assuming partition into equal sized parts) where S_i is the number of solutions in the partition given to agent i and $\sum S_i = S$ so that $\sum p_i = Np$. This gives $C^i_{\text{solve}}(k) = (1-p_i)^{k-1}$ and $C^{(N)}_{\text{solve}}(k) = \prod_{i=1}^{N}(1-p_i)^{k-1} \equiv \rho^{k-1}$. Hence the distribution of number of steps to find a solution becomes

$$P^{(N)}_{\text{solve}}(k) = \rho^{k-1}(1-\rho) \tag{11}$$

As for the nonpartitioned case, this is also a geometric distribution. With uniform steps of length $1/\lambda$, the average solution time is

$$t^N_{\text{avg}} = \frac{1}{\lambda(1-\prod(1-p_i))} = \frac{1}{\lambda(1-\rho)} \tag{12}$$

Because of the variation in the individual S_i values, $\rho = \prod_i(1-p_i)$ will usually be smaller than $(1-p)^N$ thus giving a more rapid search than the unpartitioned case.

5.4 Cooperative agents

The main question to be answered by a cooperative search theory is how to model the effect of hints. As discussed in §3.5, hints modify the behavior of the search by focusing effort in certain parts of the space. In this section we present a simple theory of the consequences of this focusing.

In general, the effectiveness of a hint will depend on what an agent has done, e.g., how good a partial state it currently has (as an extreme case, an agent finding a solution could generate a perfect hint for the others). This complicates the theory since it means agents are no longer independent and the simple relationship of Eq. (9) between the behavior of individual searches and that of the collection of agents is no longer valid. Nevertheless, we can obtain insight into the effect of hints and retain this simple relationship by treating the hints as externally given and not affected by the status of the agents' searches. This is not realistic when the quality of hints (or their ability to be usefully interpreted) depends on the state of the sending or receiving agents, but could be a useful approximation when agents are using very different strategies so that there is little or no correlation between where an agent is with respect to its own strategy and the usefulness

of a hint it generates for another agent. Alternatively, this can be viewed as an approximate "mean-field" theory in which agents are assumed to interact with hints whose effectiveness is that produced by an average agent. (Note that this doesn't necessarily mean that all hints must be the same, just that the distribution of hint effectiveness doesn't depend on the state of the sending or receiving agents.) The enhanced cooperative performance predicted by this simplified theory is observed in our experiments, suggesting that the simplified theory still captures the main phenomena.

5.4.1 Deterministic Search

In a deterministic search, the possible states are examined in some pre-specified order until a solution is found. A hint can be viewed as pruning the search space, as in the example given in §3.2. The hint effectiveness is represented numerically by the ratio of the amount of search it leaves after application to that before. Thus a perfect hint will have a hint value, or h-value, of 0 and a hint that does not move the agent closer or further from the solution has an h-value of 1. h-values greater than 1 mean that after application of the hint the agent is further away from a solution than it was before. Agents can interpret the same hint differently when they are in different parts of the search space. Thus, in general each hint will have a different h-value for each agent that uses it.

In a deterministic search method, at each step the agent will receive some number of hints (possibly zero), determine the next state to examine and test whether it is a solution. The use of hints will cause the agent to move to some different location in the space and continue its search from there, possibly restricting the states it examines to a subset of the original states. This means that, for agent i, the search remaining after step k is related to the search remaining after the previous step by

$$h_i(k)T = h_i(k-1)Th_{k\to i} - 1 \tag{13}$$

where $h_{k\to i}$ is the *net* h-value of the hints that are used during the k^{th} search step of the i^{th} agent after any previous hints have been applied. Note that $h_{k\to i} = 1$ if no hints are used. This gives

$$h_i(k)T = h_i(0)T\prod_{j=1}^{k} h_{j\to i} - \sum_{j=1}^{k}\prod_{l=j+1}^{k} h_{l\to i} \tag{14}$$

where $h_i(0)T$ is the number of steps required to find a solution without the use of hints. The process terminates when the remaining space reaches zero.

As additional hints arrive during a search, eventually they will be repeating information an agent already has and the corresponding $h_{k\to\ i}$ values will be near one, i.e., have no effect on the search. This is simply modeled by assuming there is some maximum number H of distinct hints: once an agent has received all of them, there is no new information available from additional hints. Furthermore we suppose that while these distinct hints are being received, the successive h-values are not too correlated (i.e., the ability to use a hint is not strongly related to the previous hints received). This gives $h_{k\to i} = \prod_l h_{k\to i}^{(l)}$ where $h_{k\to i}^{(l)}$ is the h-value of the l^{th} hint received during step k.

With these simplifications we can make a specific comparison between cooperative and non-cooperative searches. As an extreme contrast, suppose all the hints are available at the first step, corresponding to an extremely rapid rate of hint generation. Then we have $h_i(k)T = h_i(0)T\prod_{l=1}^{H} h_{1\to i}^{(l)} - k$ and this agent's search completes at step $k_{\text{solve}}^i = h_i(0)T\prod_{l=1}^{H} h_{1\to i}^{(l)}$. When the number of steps involved is large and they occur at a rate λ, this leads to a lognormal distribution of the number of steps required for the agents to complete their searches [14]. Specifically, we take the logarithm of this expression for k_{solve}^i. If the individual h-values aren't too correlated and H is large, then the Central Limit Theorem applies to $\sum \ln h_{1\to i}^{(l)}$ to give a normal distribution with mean $H\mu$ and variance $H\sigma^2$ where μ and σ are, respectively, the mean and standard deviation of the logarithms of the h-values. For simplicity, we suppose these moments are the same for each agent.

Finally, this needs to be related to the distribution of solution times. (Relating a product of steps to completion time is a subtle problem often overlooked, as in Ref. [24].) In our case, if the steps take place at a uniform rate λ, the solution times for the individual agents are then determined by

$$P_{\text{solve}}^i(t) \approx \Lambda\left(H\mu + \ln\frac{h_i(0)T}{\lambda}, \sqrt{H}\sigma, t\right) \tag{15}$$

where $\Lambda(m,s,t) = \frac{1}{st\sqrt{2\pi}} e^{-\frac{(\ln t - m)^2}{2s^2}}$ is the lognormal distribution [1]. If processing hints incurs an extra cost c compared to the noncooperative case, then this distribution is changed by reducing the effective rate of the steps, i.e., $\lambda \to \frac{\lambda}{1+c}$ which shifts the distribution to larger times. Finally, the distribution of time to first solution for N agents is then obtained from this using Eq. (9). Note that the highest speed that can be achieved in the process is one step, which corresponds to the situation where an agent starts with the state containing the answer. This provides an effective cutoff to the lognormal distribution for high performers, eliminating the familiar anomalies of the moments [23].

By comparison, without hints agent i will require $h_i(0)T$ steps to find a solution and the collection of non-cooperating agents will require $\min_i h_i(0)T$ steps. Because the lognormal distribution has an enhanced tail, it will generally give much higher performance (measured by the time to find a solution) for a large group of agents. In the more realistic case in which hints arrive more slowly, Eq. (14), gives a combination of multiplicative and additive contributions to the solution time so the distribution will not be strictly lognormal. However numerical evaluation of the equation does show a similar distribution with an enhanced high performance tail.

A remaining question is how the number of effectively independent hints H relates to the number of agents N. At one extreme, if all agents use exactly the same search method then they will all generate the same hints and H will be independent of the number of agents. In such a case, there is nothing to be gained from sharing information. A constant value of H also arises if agents use more sophisticated search methods which prune regions of the search space (e.g., our experiments in which each agent was restricted to post and read hints from its own blackboard only) but don't share information. The more interesting case occurs when the agents have different search strategies or expertise and can produce and use hints in ways that others cannot. Such a case offers the possibility of a large cooperative enhancement in which H grows with N when hints are shared. Thus cooperation provides a benefit not only through multiplicative pruning of the search space (which sophisticated individual search methods can also provide) but also by increasing the range of hints that each agent can apply.

This theory can also be used to give a simple criterion for when cooperation is expected to be more effective than independent search. Suppose processing a hint during a step incurs an additional cost c (i.e., the additional time required to complete the step due to interpreting a hint). From Eq. (13), using one hint at step k will incur a total cost of $1+c$ and reduce the remaining search space by $1 + h_i(k-1)T(1 - h^{(1)}_{k\to i})$ whereas independent search will incur a unit cost and reduce the space by one state. Thus the expected benefit of cooperation for a single agent outweighs the cost when

$$h_i(k-1)T(1 - \langle h \rangle) > c \tag{16}$$

where $\langle h \rangle$ is the expected hint effectiveness for the agent at step k and $h_i(k-1)T$ is the remaining search at the start of the step. In practice, neither of these quantities is known precisely during the search so estimates must be used. Nevertheless, this gives qualitative insight into the conditions under which a cooperative approach would be beneficial: hints on average must be helpful in pruning, i.e., $\langle h \rangle < 1$, and the remaining search space must be large so that pruning by the hints will eliminate many states.

This criterion essentially describes when cooperation is expected to be beneficial for an individual agent. However, when a group of agents searches and only the first solution found by any agent is desired (or best solution in a given amount of time for a satisficing search), this criterion for cooperation is too strict. That is, cooperation could be beneficial for the group as a whole even if many or most of the agents go more slowly due to misleading hints: such agents could still help by generating a few good hints which greatly increase the speed of the few high performers.

5.4.2 Probabilistic Search

With probabilistic search methods the amount of search remaining after reaching a particular state will vary depending on subsequent random choices. Thus the description of hints as pruning the remaining search space for a deterministic method must be generalized. Hints can still be viewed as focusing the search onto certain subsets of the search space. Good hints will then decrease the expected time to find a solution, even though there will be fluctuations due to the probabilistic search method. Misleading hints, on the other hand, will increase the expected time to find a solution. This suggests that an appropriate generalization of

Eq. (13) is

$$k^i_{\rm avg}(k) = k^i_{\rm avg}(k-1)h_{k\to i} - \delta \tag{17}$$

where $k^i_{\rm avg}(k)$ is the expected number of steps to a solution after step k for agent i (assuming no more hints are received) and δ is one if the search is done without replacement (states never examined more than once) and zero if states can be reexamined. In this context, the h-values characterize the change in *expected* number of steps to a solution produced by a hint.

This multiplicative process gives behavior qualitatively similar to the deterministic theory described above. For example, consider random generate and test with replacement (so $\delta = 0$). Then we have $k^i_{\rm avg}(k) = k^i_{\rm avg}(0)\prod_{j=1}^{k} h_{j\to i}$. With very rapid generation of hints this becomes $k^i_{\rm avg}(k) = k^i_{\rm avg}(0)\prod_{l=1}^{H} h^{(l)}_{1\to i}$ independent of k and thus the expected number of steps to reach a solution is distributed lognormally among the agents.

Using Eq. (2), for simple random generate and test, the expected number of steps to a solution for agent i is related to the probability for selecting a solution at each step by

$$k^i_{\rm avg}(k) = \frac{1}{P^i_{\rm success}(k)} \tag{18}$$

In particular, if the agents search in the full, unpartitioned space, this gives $k^i_{\rm avg}(0) = 1/p = T/S$. Thus for this type of search the individual $P^i_{\rm success}(k)$ are also lognormally distributed and independent of k, when the hints arrive very rapidly and so are available at the first step.

To compare the predictions of this theory with the behavior of noncooperative search, as well as our experimental results, we evaluated the expected time to first solution for a group of N agents using random generate and test. Specifically, for the noncooperative searches, Eq. (12) was used to evaluate the average solution time (with $\lambda = 1$) for N agents searching the full space (so $p_i = S/T$) as well as a partitioned space (so $p_i \approx S_i/(T/N)$, the approximation being due to the fact that the partitions were not exactly uniform when the number of agents does not evenly divide the number of search states). The cooperative results were obtained by using Eq. (12) with the values of p_i drawn from a lognormal distribution, but constrained to be no larger than one. This corresponds to the behavior expected when all the hints are available at the first step. Further, we supposed that additional agents introduce extra diversity so H increases with N. Such a comparison is shown in Fig. 11. Note the super-linear speedup due to cooperation when agents introduce additional diversity. By comparison, both the partitioned and unpartitioned noncooperative methods given linear speedup. All methods eventually saturate at a maximum speed of one, well beyond the range of agents shown in the figure.

These examples have considered the case where all the hints are available at the beginning of the search. A more realistic case is to have agents search while they receive hints. From Eq. (18), this corresponds to the values of $P^i_{\rm success}(k)$ changing by a multiplicative factor when hints are received during the search. The rate of hint arrival can range from very fast so all hints are available before the first step, to very slow so the problem is solved before any hints arrive (effectively a noncooperative case). As a specific example, we suppose the hints arrive probabilistically (e.g., at each step, for each agent, each unused hint has a probability p_h to arrive). Note that $p_h = 0$ corresponds to the case of no hints, and $p_h = 1$ has all the hints arriving at the first step, as assumed in the previous example. This description of hint arrivals, together with a characterization of their effectiveness (i.e., values for μ and σ) characterize the values of $P_{\rm success}(k)$. With Eq. (1) and (9), this gives the distribution of number of steps to reach a solution, $P^{(N)}_{\rm solve}(k)$. In particular, we have

$$C^{(N)}_{\rm solve}(k) = \prod_{i=1}^{N}\prod_{m=1}^{k-1}\left(1 - P^i_{\rm success}(m)\right) \tag{19}$$

This distribution, shown in Fig. 12, is geometric for the non-cooperating case and monotonic but not geometric when all hints are available at the start. When the hints arrive during the search, we see a transition between these behaviors: initially the solution probability is near that of the non-cooperating case (before many hints are available, the agents act effectively as if they do not cooperate); then increases toward that of having all the hints.

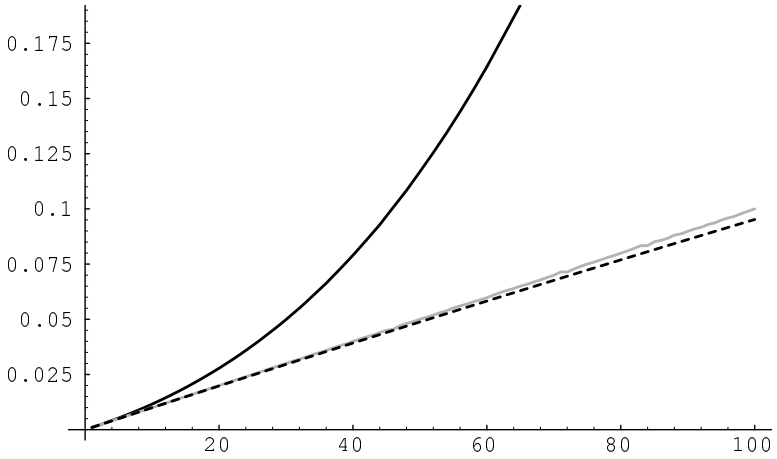

Figure 11: Search speed (defined as the reciprocal of the average time to find a solution) vs. N for cooperative and non-cooperative search. In all cases the agents use random generate and test with replacement. For the cooperative case, the number of diverse hints equals the number of agents, $H = N$, and the moments of the hint effectiveness values were $\mu = 0$, $\sigma = 0.2$. The search space consisted of $S = 1$ and $T = 1000$. The black curve is the cooperative case, the gray curve is non-cooperative partitioned search, and the dashed curve is unpartitioned non-cooperative search.

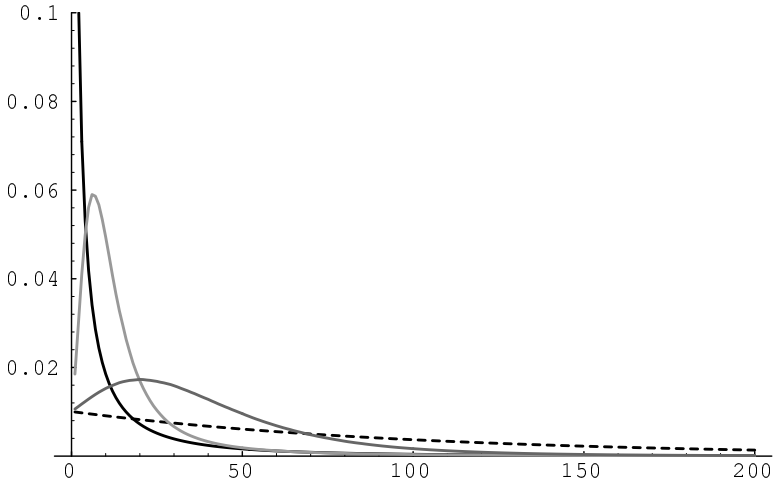

Figure 12: Distribution of first finishers, $P^{(N)}_{\text{solve}}(k)$ vs. step k for the case of $N = 10$, $H = 10$, $S = 1$, $T = 1000$ and with the hints characterized by $\mu = 0$, $\sigma = 1$. The solid black curve has $p_h = 1$, i.e., all hints available at the start of the problem. The dashed curve has $p_h = 0$, corresponding to no hints. The gray curves are intermediate cases with $p_h = 0.1$ (light gray) and $p_h = 0.01$ (dark gray) which correspond to different hint arrival rates. Note that the cases where the hints arrive over time are unimodal corresponding qualitatively to the experimental first finisher distribution in Fig. 3.

5.5 Dynamic cooperation

The above discussion presented simple criteria for when cooperation is beneficial. They involve the cost to process hints and the effectiveness of the hints from various agents. In realistic searches, these may vary with time, e.g., as better hints become available during the search. This implies that the criteria will change dynamically so that the relative benefit of cooperation will change. Exploiting this possibility requires either a central decision to tell the agents when to cooperate or allowing the agents themselves to decide. Centralized decisions are simpler to implement but are unlikely to be able to rapidly exploit local opportunities in a distributed environment. For instance, in a search, agents that appear to be near solutions (e.g., have many correct letters) may be better off by ignoring possibly distracting hints while those

encountering many deadends may improve by accepting hints from more successful agents.

If decisions are made locally, agents may choose to cooperate based on whether they perceive some net benefit to their own search. Ideally, this benefit would be designed to reflect the expected contribution to the overall progress of the group of agents. Unfortunately, such global information may not be available to the agents in a timely manner or may be difficult to analyze. Thus in such communities one is often forced to rely on simpler estimates of the local benefit to each agent. Examples of this kind of distributed decision-making occur in human market economies and some proposals for distributed computation [26]. This then raises the interesting issue of collective action problems which are present whenever an agent can benefit from the outcome of a group activity without incurring the cost associated with its own participation. Since all agents can opt to do so, this free riding mechanism implies that pervasive rationality will always be in conflict with the collective good, thus preventing any group of agents from spontaneously engaging in any cooperative action. This is the case even in situations where the problem to be solved involves interaction over prolonged periods of time. Analysis of this paradox has led to the conclusion that only small groups can sustain collaboration [11]. In order to secure cooperation in large settings one therefore needs a federated structure whereby the group is broken into smaller committees connected by some hierarchical structure of monitoring and communication.

6 Discussion

In this paper we have shown how cooperating agents working towards the solution of a constraint satisfaction problem can lead to a marked increase in the speed with which they solve it compared to their working in isolation. In particular, we showed how a diversity of cooperating agents can lead to a super-linear speed-up with agent number in the time to solution. This was compared to the case of non-diverse and non-cooperating agents where linear speed-up was observed.

The effects of diversity are especially important for the fastest and slowest agents. For a sufficiently large number of agents, the group with the highest diversity was able to solve the problem first. Interestingly, high diversity not only leads to very high performers but to very low ones as well.

This work suggests an alternative to the current mode of constructing task-specific computer programs that deal with constraint satisfaction problems. Rather than spending all the effort in developing a monolithic program or perfect heuristic, it may be better to have a set of relatively simple cooperating processes work concurrently on the problem while communicating their partial results. This would imply the use of "hint engineers" for coupling previously disjoint programs into interacting systems that are able to make use of each others (imperfect) knowledge. Although we have tested our model against only one type of constraint satisfaction problem, we believe that there are more general problem solving settings where we expect cooperation to be beneficial.

In our cryptarithmetic implementation we defined hints in terms of information that moved the agents toward a region of the space that could have a solution. Another possibility is for hints to contain information that tends to move away from regions that can have no solutions. Our theoretical derivation does not depend on which approach is used, only that some part of the search space is pruned, or is more likely to contain a solution. More generally, any search algorithm that agents may use will have parameters that will have an impact on the effectiveness of cooperation. An intrinsic advantage of cooperative search over purely algorithmic search is the former's ability to make large jumps in the search space by opportunistically utilizing shared information. Although in some cases these jumps will prove detrimental, the ability to make large excursions around the search space based on shared information is generally advantageous for at least a few agents. Another consideration is when are the hints most useful for problem solving. At the beginning of a problem the hints provide crucial information for starting the agents off on a plausible course, but will usually be fairly nonspecific. Near the end of the problem however, there are likely to be many detailed hints but also of less relevance to the agents since they may have already discovered that information themselves. This suggests that typical cooperative searches will both start and end with agents primarily working on their own and that the main benefit of exchanging hints will occur in the middle of the search.

Because our results provide a quantitative relationship between performance, number of agents, and the ability of agents to make use of diverse hints, this new methodology may be particularly useful in areas of artificial intelligence such as design, qualitative reasoning, truth maintenance systems and machine learning.

Researchers in these areas are just starting to consider the benefits brought about by massive parallelism and concurrency.

Our discussion focused on the benefits of cooperation among agents that search the space differently (due to different initial states and different hints selected from the blackboard). The most natural way to think of this is a collection of independent processes, possibly running on separate processors. However, it is always possible to have a single computational process that, in effect, multiplexes among the procedures followed by this diverse set of agents. In this way, a single agent could also obtain the benefit of cooperation discussed here. This ability of one computational process to emulate a collection of other processes is quite distinct from other cases of cooperation, e.g., human societies, where individuals have differing skills that are not easily transferred to others.

For computational processes, the issue is not so much cooperation as to what extent partial results from a range of methods are used. That is, the cooperative speed-up is due to applying a diverse set of methods to a problem, each of which fails under different conditions and can sometimes benefit from information provided by other methods. We have shown that even when the individual methods are imperfect and the information exchanged is not always correct, the overall benefit can be very large. This can be contrasted with the usual emphasis on improving individual methods. It is presently an open question as to how to exploit this diversity explicitly in problem solving or to know how to estimate it *a priori*.

Thus we see the emphasis here is on the diversity of methods available to approach a problem, as well as their ability to sometimes benefit from the partial successes of each other. In effect, we observe a huge speed-up as diversity and effectiveness of the exchange of information increases. This contrasts with discussions of the speed-up due to parallel processors in which the speed of a fixed algorithm is measured when running in a number of processors. It is well known from Amdahl's law that the speedup cannot be faster than linear in the number of processors, and is usually less due to communication overheads. There is no such restriction on the speed-up obtainable when additional diversity is introduced, either in the form of new processes running on additional processors, or by having a single algorithm incorporate additional methods.

One of the questions to be addressed is, how "dumb" agents can be yet still benefit from cooperation? If agents are too simple then they will not be able to utilize the information from other agents and thus not benefit from cooperation. For example, a simple random generate and test search that replaces all letter assignments at each step will immediately lose whatever potential benefit a hint gave it. As we saw, simple search methods with some memory can benefit from hints. This issue arises in non-computational examples as well. At the one end of the spectrum consider insect societies, where individuals are very poor problem solvers but when cooperating they are able to solve very complex tasks such as nest building, defense, and food gathering. At the other end of the spectrum the agents of a scientific community are very good at individual problem solving. This is greatly amplified by cooperation through publications and collaborations. Because cooperation appears to improve problem solving at both extremes of individual agent ability, from simple ones as in an insect society to scientists in a technical community, the benefits of cooperation are widely applicable. The agents we considered in our experiments were capable of only very simple interactions and behaviors and yet achieved considerable performance improvement versus isolated agents.

In closing, we comment on the implications of these results for social and biological organizations. In spite of the fact that we studied extremely simple agents (with no learning capabilities, and no specialization) these results may also be relevant to the more complex agents that make human organizations. We base our belief on the established fact that measures of productivity in the scientific community [24], individual problem-solving [4], as well as income distributions in a variety of economies, exhibit a lognormal distribution [1]. Although those distributions have been accounted for in terms of individual probabilities for a given agent to either obtain a certain publishable result or accrue a net worth, cooperative efforts as studied in this paper do give rise to the same performance characteristics, thus suggesting an alternative explanation.

References

[1] J. Aitchison and J. A. C. Brown. *The Log-normal Distribution*. Cambridge University Press, Cambridge, 1957.

[2] Gene M. Amdahl. Validity of a single processor approach to achieving large scale computing capabilities. In *Proc. of the AFIPS Computing Conference 30*, 1967.

[3] R. Axelrod and W. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, March 1981.

[4] David S. Bree. The distribution of problem-solving times: An examination of the stages model. *Br. J. Math. Stat. Psychol.*, 28:177–200, 1975.

[5] Kathleen Carley, Johan Kjaer-Hansen, Michael Prietula, and Allen Newell. Plural-soar: A prolegomenom to artificial agents and organizational behavior. In Michael Masuch and Massimo Warglien, editors, *Artificial Intelligence in Organization and Management Theory*, chapter 4, pages 87–118. North-Holland, Amsterdam, 1992.

[6] Scott H. Clearwater, Bernardo A. Huberman, and Tad Hogg. Cooperative solution of constraint satisfaction problems. *Science*, 254:1181–1183, 1991.

[7] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[8] Peter J. Denning and Walter F. Tichy. Highly parallel computation. *Science*, 250(4985):1217–1222, November 30 1990.

[9] R. Engelmore and T. Morgan. *Blackboard Systems*. Addison Wesley, Reading, MA, 1988.

[10] Les Gasser and Michael N. Huhns, editors. *Distributed Artificial Intelligence*, volume 2. Morgan Kaufmann, Menlo Park, CA, 1989.

[11] Natalie Glance and Bernardo Huberman. Expectations, uncertainty and free rider problems. Technical Report to appear, Xerox PARC, Palo Alto, CA, 1992.

[12] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, NY, 1989.

[13] B. A. Huberman and T. Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33:155–171, 1987.

[14] Bernardo A. Huberman. The performance of cooperative processes. *Physica D*, 42:38–47, 1990.

[15] W. A. Kornfeld and C. E. Hewitt. The scientific community metaphor. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11:24–33, 1981.

[16] William A. Kornfeld. The use of parallelism to implement heuristic search. Technical Report 627, MIT AI Lab, 1981.

[17] William A. Kornfeld. Combinatorially implosive algorithms. *Communications of the ACM*, 25(10):734–738, October 1982.

[18] Jean-Louis Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.

[19] Eric Lumer and Bernardo Huberman. Dynamics of resource allocation in distributed systems. Technical Report P90-00007, Xerox Palo Alto Research Center, Palo Alto, CA, 1990.

[20] A. K. Mackworth. Constraint satisfaction. In S. Shapiro and D. Eckroth, editors, *Encyclopedia of A.I.*, pages 205–211. John Wiley and Sons, 1987.

[21] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[22] H. Penny Nii, Nelleke Aiello, and James Rice. Experiments on Cage and Poligon: Measuring the performance of parallel blackboard systems. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence*, volume 2, pages 319–383. Morgan Kaufmann, San Mateo, CA, 1989.

[23] S. Redner. Random multiplicative processes: An elementary tutorial. *Am. J. Phys.*, 58(3):267–273, March 1990.

[24] William Shockley. On the statistics of individual variations of productivity in research laboratories. *Proc. of the IRE*, 45:279–290, 1957.

[25] H. Simon. *The Sciences of the Artificial*. M.I.T. Press, Cambridge, MA, 1969.

[26] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffery O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering*, 18(2):103–117, February 1992.

[27] E. O. Wilson. *The Insect Societies*. Harvard University Press, Cambridge, 1971.