Asynchronous Teams: Near-Scale-Effective Organizations for Distributed, Computer-Based Agents

Sarosh Talukdar, Andrew Gove and Pedro de Souza

EDRC 05-94-95

ASYNCHRONOUS TEAMS: NEAR-SCALE-EFFECTIVE ORGANIZATIONS FOR DISTRIBUTED, COMPUTER-BASED AGENTS

Sarosh Talukdar Andrew Gove Pedro de Souza
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT
An asynchronous team (A-Team) is a strongly cyclic computational network of autonomous agents and shared memories. Results circulate through this network. Agents cooperate by working on one another's results. There is a growing body of evidence that such networks are extremely effective in solving optimization problems. Moreover, they seem to benefit from increases in scale: big networks are more effective than small ones. This article explains why.

INTRODUCTION
Many algorithms are available for optimization and constraint satisfaction. Some, such as branch-and-bound, are rigorous, others, such as genetic algorithms, are heuristic. However, none of the available algorithms is entirely suitable for many, if not most, practical problems. The rigorous algorithms tend to be too slow, the heuristics, too unreliable.

For several years we have been experimenting with a prescription for assembling algorithms into teams so they can cooperate. We arrived at this prescription by combining features from a number of natural and synthetic systems, particularly: insect societies [1], cellular communities [2], genetic algorithms [3], blackboards [4], simulated annealing [5] and tabu search [6]. The resulting prescription produces strongly cyclic computational networks called asynchronous teams (A-Teams). Results are circulated through these networks by software agents. The number of agents can be made arbitrarily large. Each agent is completely autonomous (it decides what it is going to do and when, just as do the members of an in$ect society). Results that are not being worked on by the agents accumulate in shared memories Gust as happens in blackboards) to form populations (like those used in genetic algorithms). Destruction (the elimination of weak results), inhibition (the avoidance of certain classes of results)

and chance play key roles in determining what happens to the populations (just as they do in natural systems and to a lesser extent, in some synthetic systems).

While all the features of A-Teams can be found in biological systems, there is one that is rare in synthetic systems for computational problems. This feature is the complete absence of any centralized planning or coordination. Agents in an A-Team work without supervision and cooperate by modifying one another's results. This mode of cooperation is asynchronous: agents choose which results they will work on and when. No agent is ever forced to wait for results from another. Rather, all the agents can work in parallel all the time. In contrast, other synthetic systems invariably have some scheme to force at least a partial order on the activities of their computational modules.

We have applied A-Teams to a number of large and difficult problems including traveling salesman problems [141, sets of nonlinear equations [7], high-rise building design [8], reconfigurable robot design [9], diagnosis of faults in electric networks [10], control of electric networks [11], job-shop-scheduling [16], protein structure analysis [17], and train-scheduling [15]. Other groups, who have borrowed the A-team technology from us, have applied it to steel-mill scheduling, paper-mill scheduling and constraint satisfaction [18]. In all these cases, A-Teams have proved to be remarkably effective. They find better solutions more quickly than can any of their constituent algorithms. Indeed, A-Teams seem to provide some benefits-of-scale: the more algorithms (each formed into an autonomous agent), the better the results. Why is this? The following material provides some clues.

A PRESCRIPTION FOR DESIGNING A-TEAMS

In other articles, we have defined A-Teams in terms of their features. Here, we will define them constructively. Specifically, we will give a seven step prescription for producing an A-Team and illustrate it with one of our earlier applications, the traveling salesman problem (TSP).

Step 1: Choose a difficult computational problem

The TSP is an optimization problem that can be simply stated, is extremely difficult to solve and has a wide range of practical uses. The problem itself is: given m cities and their inter-city distances, find the shortest tour of the cities. (A tour is a closed path that goes through every city.) The number of distinct tours (there are(m-1)! /2 of them) grows so rapidly with m that it is impractical to conduct an exhaustive search for the shortest tour, even when there are as few as 30 cities. Practical problems often have hundreds or even thousands of cities.

Step 2. Decompose the problem into related-problems

One of the many ways in which a TSP can be decomposed is into the following three related-problems: finding good complete tours, finding good partial tours, and finding

good 1-trees. (A.1-tree is a path that goes through all the cities but is not quite closed. 1-trees are useful in determining lower bounds for TSPs.)

Step 3. Assign a memory to each related-problem

The purpose of this memory is to hold a population of trial-solutions to its problem. To complete the step, an upper bound on the size of the population must be set and a representation for each of its members designed. For the TSP we set the upper bound at 500. We chose ordered lists of cities as our representation for complete tours, partial tours and 1-trees. For instance, {Atlanta, Boston, Raleigh, Pittsburgh} means a tour that goes from Atlanta to Boston to Raleigh to Pittsburgh, and then back to Atlanta.

Step 4. Select a representative sample of the available algorithms for each related-problem

The TSP is an NP-hard problem, which means that rigorous algorithms that can solve it in reasonable (polynomial) amounts of time are unknown. However, there are literally hundreds of relatively fast heuristics for generating and refining sub-optimal tours. For instance, "arbitrary insertion" (At) is a simple heuristic that creates complete tours from partial tours as follows [14]: a) arbitrarily pick a city from among those not included in the partial tour, b) insert this city into the partial tour at the cheapest available site, and c) repeat till a complete tour is obtained. There are also some very complicated and much more powerful heuristics. One of the best is the Lin-Kernighan (LK) algorithm [12].

The sample of algorithms we chose is given in Fig. 1.

Step 5. Form each algorithm into an autonomous agent

Think of an agent as operating between two worlds: one it perceives (its input-world), the other it affects (its output-world). Then any agent can be thought to consist of two components:

- an operator that transforms stimuli from the input-world into effects on the output-world, and
- a control system that chooses the stimuli for the operator and schedules its activities.

For an agent composed purely of software, the input- and output-worlds can always be modeled as memories (that may be shared and may overlap) [19]. The operator can be modeled as a mapping from an "in-space" (the set of all the objects that can be stored in the input-memory) to an "out-space" (the set of all the objects that can be stored in the output-memory). The control system can be thought to consist of three subsystems: sensors to examine the contents of the input-memory, selectors to choose objects from these contents, and schedulers to decide when the operator wilt work on the selected objects.

If the control system is completely self-contained, that is, if the agent accepts no instructions on what to do from the outside, then the agent is said to be autonomous.

Thus, to convert an algorithm into an autonomous agent one needs to designate its input and output memories, provide communications with these memories, and add control programs for input-selection and activity-scheduling. How should this selection and scheduling be done? There are many possibilities. For the TSP, we used random selection, biased in favor of the best solutions. The scheduling strategy for each agent was to run continuously with all the processing resources made available to it.

Step 6. Interconnect the agents and memories into a strongly cyclic network in which resultscancirculateindefinitely

It is helpful to visualize networks of agents and memories as directed graphs in which nodes denote memories and arcs denote agents, as in Fig 2. Such graphs are called data flows. A data flow is strongly cyclic if most of its agents work in closed loops, it is often necessary to create a few new agents to close loops and make a data flow strongly cyclic. In the case of the TSP, there are few standard algorithms for deconstruction (breaking complete tours into good partial tours). Therefore, to close the loop through the complete and partial tour memories (Ftg.2.b) we developed a deconstruction algorithm of our own. Another type of agent, called a destructive agent, must also be developed. Destructive agents eliminate poor solutions and make room for new solutions to be added. Each memory must have at least one. Otherwise, the memory will become clogged with solutions and circulation will cease. In the TSP, the destructive agents were designed to eliminate randomly, using a monotonic distribution (the probability of elimination was made to increase monotonically from zero, for the best solution in the memory, to a maximum, for the worst solution).

Step 7. Seed the memories with starting solutions and run the network. If the solutions are overly slow in converging to acceptable values, repeat from step 2, adding more agentsand/ormemories

Fig. 2 shows some networks for the TSP and Fig. 3 shows the results they produce for a number of difficult problem-instances. All of these results are far better than can be produced by LK-the best of the algorithms used in the team-when it is working alone. Notice that there is an improvement in the quality of the results and often, in the speed with which they are produced, as the networks increase in size. This is so even though

all the agents must share just one computer (a Dec 5000). When each agent was given its own computer, there was a profound increase in speed for the larger networks, 3.c and 3.d. We have observed such benefits-of-scale in all the cases we have tackled. More specifically, the performance of the team invariably improves as computers are added until each agent has all the computer resources it can use. Performance also improves with some, but not all, expansions of the sets of agents and memories.

MODELS OF COOPERATION AND EFFECTIVENESS

Any computer-based oranization, if it works at all, will work better if its computers are made faster or are better utilized. Here we will examine some features of organizations and cooperation that are computer-independent.

Cooperation can be defined in a variety of ways. We use a very broad definition: agents cooperate when they exchange data, regardless of whether the exchanges are profitable or not. If the agents are computer-based, then every way in which they can exchange data can be modeled by a graph of the sort that has been called a data-flow [19]. In other words, the space of all data-flows captures all the different ways in which computer-based agents can cooperate. In still other words, given a set of agents, the problem of designing an effective way for them to cooperate reduces to a search through the space of all data flows for an effective data flow.

Terminology

How can effectiveness be measured? Consider any node of a data-flow. This node represents a memory. The memory contains a population of competing solutions to P, a problem, that is related in some way to OP, the overall problem to be solved. If P is the same as OP, the memory is called a primary memory. Let S be the space (set) of all possible solutions (including bad ones) to P, and let G, called the goal space, be the subset of S that contains all the acceptable solutions to P. Assume that G is non-empty (the problem has at least one acceptable solution). Suppose that the memory is large enough to hold a population of N solutions. If the node is to be effective, this population must evolve a non-empty intersection with G (that is, at least one of the solutions in the memory must evolve into an acceptable solution). We measure the effectiveness of a memory by the expected speed (the reciprocal of the expected time) of this evolution, and the effectiveness of the entire network by the effectiveness of its fastest primary memory. Note that by this definition, effectiveness is problem-specific. An organization's effectiveness will vary from one problem to another.

The evolution of a population of solutions in a memory is, of course, determined by the agents that act on (write to) the memory (Fig. 4). These agents may be divided into two sets: constructive agents that add new members to the population or modify existing members, and destructive agents that delete members from the population.

Visualize the evolution of a population as a tangle of paths that are traced through S. Each of these paths is a sequence of points: {xo, x-|, —, XL}. The starting point, xo, is one of the "seeds[11] placed in the memory before any agents were activated. Each succeeding point, xj, can be thought of as being developed from its immediate predecessor, xj-i, by a constructive agent. (Sometimes the predecessor may neither be obvious nor unique. In such cases, a predecessor may be chosen arbitrarily without affecting the argument we are about to make.) The path is successful if it reaches the goal space, that is, if XL e G.

Successful paths are not unique: if there is one path from a starting point to G, then inevitably, there are many. Some of these paths can take much longer to develop than others. The task of the destructive agents is to terminate paths that are likely to be overly long, before too much time has been wasted on their development.

Let xo be any member of the initial population of solutions in the memory and let C be the set of constructive agents that act on the memory. Which of these constructive agents are needed to forge the shortest path from xo to the goal space? In what order should they be invoked? How much time will they take? To examine these very difficult questions, we will use a distance metric for S that is asymmetric and dependent on C. Specifically, we define the distance from one point to another as the minimum number of constructive operations needed to get from the first point to the second.

A Theorem on Expected Path Length

Let:

S be partitioned into disjoint regions: {So, Si,—, S^}, such that all the points in Sn are at the same distance, n, from G (Fig. 5);

UK be the subset of S containing the regions So through SK, that is, UK contains all the points that are K operations or less from G;

Rn be the residue of Sn, that is, the fraction of points in UK that are at distances of n or greater from G. In other words:

$$Rn = \hat{}j_{=n \cdot >K} [\|Sj\| / \|UK\|] \bullet$$ where II.II is a norm of set-mass;

e = 1/(p-q), where e is called the selection error, p is the probability that the next agent selected in the development of a path will move the path closer to G, q is the probability that it will move the path further from G, and (1-p-q) is the probability that it will leave the path at the same distance from G.

If:

- the destructive agents make the portion of S that is outside UK completely inaccessible, preventing paths in UK from ever leaving it;
- N starting points are randomly chosen from UK;

- the best (closest to G) of these points is identified and a path from it to G is developed by the sequential application of constructive agents,
- e is constant and non-negative at every stage of this development;

then:

Rn = 0 if and only if Sn = Sn+1 = — = SK = 0 (1)

R1. R2.-~. ^{R}K decrease monotonically as K decreases (2)

R2. R3,—. RK decrease monotonically as the variety of constructive skills in C increases, that is, as the number of agents in C increases (3)

$$Lmin = Z_{n=1->K} [R_n]^N \quad (4)$$

Lexp = s-L-min (5)

where Lmin is the distance of the best starting point from G, and Lexp is the expected length of the path from this point to G. Results (1) and (2) are obvious from the definition of Rn. The other results are proved in the Appendix.

Comments

Expressions (1) - (5) provide a model of those parts of an organization that are computer-independent. Their main implication is: increases in the effectiveness of an organization can be obtained by reductions in e and Lmin- How can such reductions be achieved? What else can we infer from the model?

Constructionanddestructionascomplementaryprocesses

Construction works by shrinking the outer regions of S, destruction, by making these regions inaccessible. As such, construction and destruction provide different means to the same ends: to make Lmin finite (either by making S^ empty or inaccessible), and to make Lmin small (by reducing the values of the residues, Ri,—,RK).

Results (3) and (4) imply that the more constructive-agents there are, and the greater the variety of their skills, the shorter Lmin will be.

The model does not treat destruction in enough detail to examine the effects of the numbers of destructive agents. But it is reasonable to believe that there are benefits to making these numbers large. Certainly, destructive agents can serve more functions than just making the outer regions of S inaccessible. Eliminating cyclic paths in S (Fig.

5) and keeping those portions of S that have been visited from being revisited, for instance.

Are construction and destruction equally useful? Synthetic systems often favor one over the other. For instance, all the "smarts" of a hill-climbing algorithm are in the process by which it constructs new solutions from old ones. Simulated annealing algorithms, however, concentrate their "smarts" in processes for destroying weak solutions. Natural systems are often more symmetrical, with large numbers of both constructive and destructive agents. The process of Lamelar bone growth [9], for instance, relies as much for its efficacy on cells that add bone material to surfaces where the stress is high, as it does on cells that remove bone material from surfaces where the stress is low.

Perhaps some forms of knowledge are easier to compile into constructive agents, others, into destructive agents. Records of past successes and recipes for what to do or where to look, probably fall into the former category; records of past mistakes and recipes for what not to do or where not to look, into the latter category. Or put another way, it is probably easier to eliminate the effects of some mistakes than to prevent them from happening.

The effects of population size, N

Lmin decreases as N increases because all the Rn in (4) are less than 1 in value, and therefore, $[Rn]^{N+1} < [Rnl^{N}$- The effects are most pronounced on the outer regions of S because $Rn \geq Rn+1$, and therefore, $[Rn]^{N} \geq [Rn+1]^{N}$. In other words, increases in population size benefit the initial stages of a search more they do than the final stages. When the inner regions are relatively small, populations must grow to astronomical proportions before they can affect the final stages of a search.

Selection error

For every point in S, the constructive agents can be divided into three categories: the right agents for the point (any one of which will, if applied, produce a new point that is one operation closer to G), neutral agents (which will produce new points no closer to G), and the wrong agents (which will produce points further from G). These categories are uncertain. At each step in the development of a path, it is possible that one of the neutral or wrong-agents will be selected. If the wrong agents are more likely to be selected than the right ones ($p < q$) then, the average path will never reach the goal.

Opportunities for concurrency

The assumptions under which the expression for Lexp was derived can be relaxed, and the value for Lexp reduced, by exploiting parallelism in at least two ways. The first, is to develop several paths in parallel. Not only does this eliminate the difficulty of identifying the best starting point, but a path originating from an inferior point could, through fortunate selection of agents, reach the goal space before any other. The

second, is to use several existing points, instead of just one, to develop the next point along a path. The additional information so brought into play can only help.

Delays

The total time taken to construct a path is the sum of the time that constructive agents spend actually working on this path plus the time by which they are delayed in their work. The delays are of three types: synchronization delays that occur when a constructive agent must pause in order to satisfy a synchronization or precedence constraint in the organization's control structure, communication delays that occur when a constructive agent must wait for the delivery of data it needs, and resource contention delays that occur when a constructive agent must wait for the computers if needs.

Synchronization delays disappear when the precedence constraints that cause them are relaxed. In other words, they can be switched on or off strictly by organizational changes, and in this sense, are computer-independent. For instance, the control structure of the traditional genetic algorithm requires construction and destruction to occur sequentially. Thus, all the construction must cease while the weaker trial-solutions are destroyed, and each peiod of destruction constitutes a synchronization delay. This delay would disappear if construction and destruction were allowed to proceed in parallel.

Scale-Effective Distributed Procesing

The question that we consider here is: what happens to the effectiveness of an organization as the number of its agents is increased? We assume that the organization uses a distributed network of computers. To simplify the analysis, we further assume that each constructive agent is provided with a computer for its exclusive use, so the resource contention delays are zero, and this computer is sized so every agent takes the same amount of time to perform one constructive operation. Then, for any memory in the organization:

$$TRP(C) = p.e.Lmin + T_Syn + Tcom \quad (6)$$

where RP is the problem associated with the memory, TRP(C) is the expected time to reach the goal (an acceptable solution to RP), C is the contingent of constructive agents that act on the memory, e.Lmin is the expected number of constructive operations needed to reach the goal, p is the constant time required for each constructive operation,Tsyn is the net synchronization delay, and Tcom is the net communication delay.

How can TRP(C) be made small? To reduce L_min requires the expansion of C. But increasing the number of agents can increase e, T_Syn and T_Com. (e tends to increase because the larger the number of agents, the more difficult it is to pick the right agent;

Tsyn because precedence constraints tend to form bottlenecks, especially in organizations with many supervisory layers; and Tcom because more agents inevitably require more data to be delivered over greater distances).

We say that a memory is scale-effective if Tp(C) decreases as C expands. An organization is scale effective if its fastest primary memory is scale-effective.

Human organizations tend to be "scale-ineffective." Hence the saying, "too many cooks spoil the broth." In a scale-effective organization, there cannot be too many cooks. More specifically, scale effectiveness has two consequences of great practical signifance to distributed problem-solving. First, an organization that is scale effective can accommodate an arbitrarily large number of agents, without restrictions on their size or generality; big agents can be mixed with small agents, generalists with specialists. In other words, the skills required by a problem can be encapsulated in a few large and very broad agents, or in an army of much smaller specialists. Second, the problem of improving the performance of a scale-effective organization reduces to one of finding which agents to add.

HOW A-TEAMS WORK

In an A-Team, all the agents are autonomous. Each does whatever it wants whenever it wants. In other words, cooperation (data exchanges) occur asynchronously: there are no precedence constraints and therefore, no agent can be made to wait on another. Indeed, all the agents can, if they so choose and if enough computers are available, work in parallel all the time. There are two consequences.

First, Tsyn = 0, regardless of the number of agents. Thus, a significant barrier to scale-effectiveness is removed.

Second, there can be no centralized plan for the control of the agents. Rather, agents must be self-selecting and self-scheduling. On the upside, this makes for flexibility. New agents can be added without having to modify any centralized planning system. On the downside, very little is known about designing good self-selection and scheduling techniques.

The experiments we have conducted [7]-[11], [14], [15] demonstrate that very simple self-selection heuristics are often more than adequate, at least for optimization and constraint-satisfaction problems. The essence of these heuristics is to select solutions randomly, with a bias towards good solutions for constructive agents, and bad solutions for destructive agents. "Good" and "bad" are measured in terms of the problem's criteria, that is, its objectives and constraints.

If an A-Team is implemented in the sort of distributed network of computers described earlier, The expected time in the sort of distributed network described earlier, has a

favorable form: T(C) = p.e.Lmin +Tcom. Scale effectiveness can be achieved by keeping increases in e and $T_C$om from outweighing decreases in Lmin. as C is expanded.

To illustrate, consider a train-scheduling problem [15]: given a set of tracks and a set of trains, minimize the total lateness of the trains subject to precedence and capacity (PC) constraints (such as, a train can travel only along connected rail segments) and a number of problem-specific (PS) constraints (such as, the minimum allowable separations between trains). PC constraints occur in one form or another in virtually all scheduling problems. The scheduling literature is full of generic algorithms that can be adapted to handle their variations. But the PS constraints are quite another matter. Customizing a generic algorithm to include large numbers of them, can, and usually is, prohibitively difficult. By using an A-Team one can avoid this difficulty as follows:

- Include as many generic algorithms as desired, each formed into an autonomous agent with a random selector.
- Dedicate a separate algorithm to each PS constraint. Design this specialized algorithm to eliminate violations of its constraint without concern for what happens to the other constraints. (This makes the algorithm easy to design but myopic: in meeting its constraint it may cause others to be violated.) Form the algorithm into an autonomous agent by including a selector that picks solutions with large violations of the algorithm's constraint.
- Design destructive agents that recognize and eliminate closed paths in solution space. (The specialized agents, being myopic, can easily get into cycles, with one agent undoing the efforts of another. It is easier to destroy such cycles than to prevent them from happening.)
- Design destructive agents to eliminate solutions that revisit parts of the space that have already been explored. (It is easier to interrupt revisits to prevent them from happening.)
- Design destructive agents that eliminate solutions based on their distances from the current Pareto (non-dominated) set of solutions; the greater the distance, the more likely the solution is to be eliminated.

An A-Team built along these lines is described in [15]. Tests indicate that it can improve on-time-arrivals from 75%, which is typical of current railroad performance, to over 90%, at which level railroads could compete, in terms of punctuality, with the trucking industry.

SUMMARY

Any computer-based organization for cooperative problem-solving can be modeled as a network of memories and agents. Each memory holds a population of trial-solutions to a problem that is related to the overall problem to be solved. If this related problem

is the same as the overall problem, the memory is called a primary memory. Every network must have at least one primary memory.

The effectiveness of a memory is measured by the speed with which its population evolves an acceptable solution to its problem. The effectiveness of the organization as a whole, is the effectiveness of its fastest primary memory.

The evolution (movement) of the population of solutions in any memory is determined by the population of agents that act on that memory. These agents can be of two types: constructive agents that add solutions to the population and destructive agents that delete solutions.

Expressions (1)-(5) provide a computer-independent model of the effectiveness of any memory. This model implies:

- Construction and destruction are dual processes. Weakness in one can be compensated by strengths in the other. In practice, it is advisable to allow for both: some forms of knowledge are easier to compile into constructive agents, others, into destructive agents.
- Starting from a random population of trial-solutions, Lmin, the minimum number of constructive operations necessary to develop an acceptable solution, decreases with increases in the size of the solution-population (N) and with expansions of the constructive-agent-population (C). But N is less influential than G.
- While expansions of C reduce Lmin. they do not necessarily increase overall effectiveness. This is because expansions of C can produce increases in the selection error (e) and the synchronization delay (Tsyn). both of which have an adverse influence on effectiveness.
- If each agent has its own properly sized computer, then the net effect of expanding C is captured by a quantity: $fp(C) = e.L_{m}in + Tsyn$- If an expansion of C is to be beneficial, it must cause fp(C) to decrease. An organization is said to be scale-effective if a beneficial expansion is always possible. Scale-effectiveness is a desirable property because it reduces the problem of improving organizational performance to one of finding the right agents to add. (A super-agent is a network of memories and agents.)

An A-Team is an organization in which all the agents are autonomous. Each does whatever it wants whenever it wants. As a result, synchronization delays are nonexistent and the only barrier to scale-effectiveness is keeping the selection-error from growing faster than Lmin shrinks. In other words, where the selection error can be kept from growing rapidly, an A-Team allows arbitrarily large numbers and varieties of agents to beneficially cooperate: massive agents with miniscule agents; rigorous agents with heuristics, general-purpose agents with specialists.

Empirical evidence suggests that it is fairly easy to keep the selection error suitably small, at least for optimization and constraint-satisfaction problems. More specifically, random selection seems to work adequately, provided it is biased towards the better solutions for constructive agents and the poorer solutions for destructive agents.

We suspect that the greatest benefits of A-Teams will be realized in widely distributed networks of computers where each of the agents can have its own computer, customized for its own needs.

REFERENCES

[I] G.F. Oster and E.O. Wilson, "Caste and Ecology in the Social Insects," Princeton University Press, Princeton, NJ, 1978.

[2] A. Kerr, Jr., "Subacute Bacterial Endocardites," Charles C. Thomas, Springfild, IL, 1955.

[3] "Handbook of Genetic Algorithms," edited by L. Davis, Van Nostrand Reinhold, 1991

[4] H. P. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, Parts I and II, *AI Magazine,* 7:2 and 7:3, 1986.

[5] S. Kirkpatrick, C D. Gelatt, and M.P. Cecchi, "Optimization by Simulated Annealing," *Science,* Vol. 220, Number 4598, May, 1983.

[6] F. Glover, "Tabu Search-Parts I and II," *ORSA Journal of Computing,* Vol. 1. No. 3, Summer 1989 and Vol. 2, No. 1, Winter 1990.

[7] P.S. de Souza and S.N. Talukdar, "Genetic Algorithms in Asynchronous Teams," *Proceedings of the Fourth International Conference on Genetic Algorithms,* Morgan Kaufmann, Los Altos, CA, 1991.

[8] R.W. Quadrel, "Asynchronous Design Environments: Architecture and Behavior," Ph. D. dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

[9] S. Murthy, "Synergy in cooperating agents: designing manipulators from task specifications," Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1992.

[10] C.L. Chen, "Bayesian Nets and A-Teams for Power System Fault Diagnosis," Ph. D. dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 1992.

[II] S. N. Talukdar, V.C. Ramesh, "A parallel global optimization algorithm and its application to the CCOPF problem," *Proceedings of the Power Industry Computer Applications Conference,* Phoenix, May, 1993.

[12] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research,* Vol. 21,1973, pp. 498-516.

[13] M. Held and R.M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees," *Operations Research,* Vol. 18, 1138-1162, 1970.

[14] P. de Souza, "Asynchronous Organizations for Multi-Algorithm Problems," Ph. D. dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1993.
[15] C. K. Tsen, "Solving Train Scheduling Problems Using A-Teams," Ph.D. dissertation, Electrical and Computer engineering Deoartment, CMU, Pittsburgh, 1995.
[16] S. Y. Chen, S. N. Talukdar, N. M. Sadeh, "Job-Shop-Scheduling by a Team of Asynchronous Agents," *IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control,* Chambery, France, 1993.
[17] Prptian stuff
[18] S. R. Gorti, S Htfmair, R. D. Sriram, S. Talukdar, S. Murthy, "Solving Constraint Satisfaction Problems Using A-Teams," to appear in *AI-EDAM.*
[19] S. N. Talukdar and P. S. de Souza "Insects, Fish and Computer-Based Super-Agents," *Systems and Control theory for Power Systems,* edited by Chow, Kokotovic and Thomas, Vol. 64 of the Institute of Mathematics and its Applications, Springer-Verlag, 1994.
[20] J. H. Kao, J. S. Hemmerle, F. P. Prinz, "Asynchronous-Teams Based Collision Avoidance in PAWS," EDRC Report, Carnegie mellon University, June 1995.
[21] P. Krolak and W. Felts, "A Man-Machine Approach Toward Solving the Traveling Salesman Problem," *Communications of the ACM,* Vol. 14, No. 5, May 1971.
[22] M. Grotschel, "Polyedrishe Kombinatorik and Schnittebenverfahren," Preprint No. 38, Universitat Augsburg, 1984.
[23] M. Padberg and G. Rinald, "Optimization of a 532-city Symmetric Traveling Salesman Problem," *Oerations Research Letters,* Vol. 6, No. 1, March 1987.

APPENDIX

Contracting the outer regions of S by expanding C

Let dc(x,y) be the distance from any point x e S to any other point y e S. We define dc(x,y) to be the minimum number of constructive operations needed to get from x to y. Each of these operations must be performed by an agent from C. Thus, dc(x,y) is dependent on C. Suppose that an agent containing a new constructive skill is added to C and this skill makes it possible to get from some point to another in fewer operations. Then these points will have grown closer. Thus, dC2(x,y) ^ dC1 (x,y) if C2 :DC1. In particular, dG2(x,g) ^ dd(x.g)'* C2 ^ C I, where g e G. Thus, the outer regions of S (Fig. 5) contract as C expands. In other words, the residues R2. R3»—. RK decrease as C expands.

Expected path-length from a starting point in S_n

Consider a starting point xrj in Sn. Suppose that a constructive agent works on xo to produce the point x i. Let:

p be the probability that xi is one step closer to G, that is, xi e Sn-1

r be the probability that xi is no closer to G, that is, xi e Sn

q be the probability that xi is one step further from G, that is, xi e Sn+1

where p + r + q = 1. Given these probabilities, we would like to know how many operations it would take to reach the goal. Our model is equivalent to a random walk in one dimension. Let L be the expected length (i.e. number of operations) it takes to reach S_n-1 for the first time. Once S_n-1 is reached, it will take an expected L more operations to reach S_n-2. because the situation is identical. Thus, it will take a total of nL time units to reach the goal. Here is an expression for L

L = expected number of operations to make one step of forward progress

= P*1

+ q * (operations to reach S_n-i from S_n+i)

+ r * (operations to reach S_n-1 from S_n)

= p + q(1+2L) + r(1+L)

L=1/(1-2q-r) = 1/(p-q)

Thus, starting with a point that is in Sn, the expected length of the path to G is:

Unin = n/(p-q) (i)

(Note: this expression is exact only when K, the destruction threshold, is infinite. When K is finite, the exact expression is more complicated, but the additional complications do not affect our arguments and therefore, are not included.)

Effective path-length from the best of N randomly chosen starting points
Consider a population of N starting points that are randomly chosen from the subspace U«. where U« consists of the regions So. Si, —, SK. and all points in UK are equally likely to be chosen. Let:

x be any one of the N starting points

P(n) be the probability that x e Sn,

R(n) be the probability that x € Sj, with j ≥ n

RN(n) be the probability that none of the N starting points is in any Sj, such that j < n.

Assume that: P(n) = HSnll / HUKII where 1111 is a norm of set-mass. Then:

$$R(n) = P(n) + P(n+1) + \text{—} + P(K),$$

$$R_N(n) = [R(n)]^N, \text{ and}$$

$$PN(n) = RN(n) - RN(n+1)$$

Le :
xbest be the closest of the starting points to G
PM(n) be the probability that xbest^e ^s n
Lmin be the expected distance of xbest from G
Then:

$$Lmin = £_{n=0.>K}\ n.PN(n)$$

$$= £_{nss0\cdot>K} n.[RN(n) - RN(n+1)l$$

$$= \sum_{n=0->K} n.R_N(n) - \sum_{n=0->K} n.R_N(n+1)$$

$$= \sum_{n=1->K} n.R_N(n) - \sum_{n=1->K} (n-1).R_N(n)$$

$$= \sum_{n=1->K} R_N(n)$$

$$= \sum_{n=1->K} [R(n)]^N \qquad \text{(ii)}$$

Combining (i) and (ii) gives:

$$Uxp = e.\ Lmin - e. \sum_{n=1->K} [R(n)]^N$$

where Uxp is the expected length of the path from xbest to G. and e - 1/(p-q)-

LK • Lin-Kernighan, one of the most powerful and most compUcated algorithms available [12]

CLK : a simplified version of LK [14]

OR : **Or-Opt, a moderately complicated algorithm [14]**

AI : **Arbitrary Insertion, a very short and simple algorithm [14]**

HK : **Held-Karp, an algorithm for converting tours to 1-Trees [13]**

Dec : **a deconstructor that produces a partial tour from the common edges of two tours [14]**

MI : **a mixing algorithm that combines two tours to get one [14]**

TM : **a mixing algorithm that combines a tour with a 1-Tree to give a new tour [14]**

Fig. ***1 A Sampling of Heuristics for Traveling Salesman Problems.***

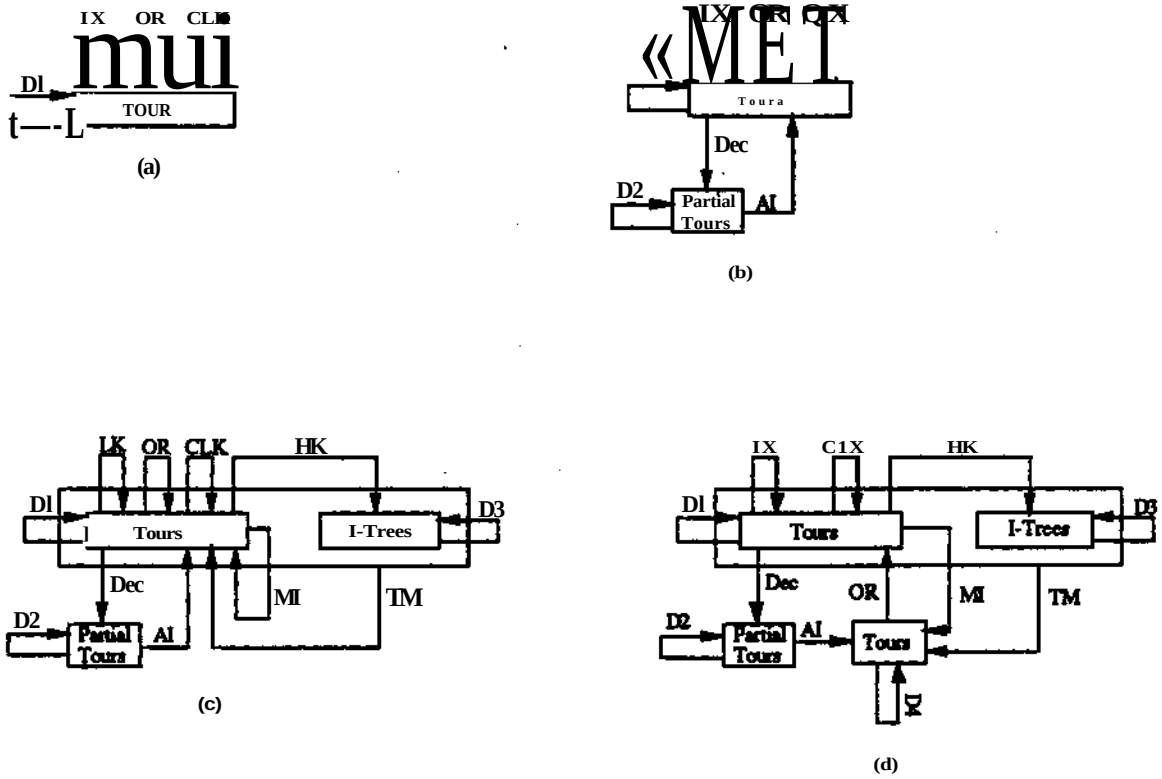

Fig. 2. Four data-flows for the traveling salesman problem. D1-D4 are destructive agents: they eliminate members from solution-populations. The other agents are constructive (they add members to solution-populations) and are based on the algorithms given in Fig. 1.

Dataflow (see Fig. 2 for details)	Distance from the optimum solution/computation time for the following problems:			
	Krolak24 100 cities [21]	LK318 318 cities [14]	PCB442 442 cities [21]	ATT 532 532 cities [23]
(a)	0% / 35 sees.	1.27% / 2.9 hrs.	1.20% / 4.2 hrs.	0.87%/7.5 hrs.
(b)	0%/39 sees.	1.13% / 2.4 hrs.	0.89% / 3 hrs.	0.47%/6.8 hrs.
(c)	0% / 39 sees.	0.06% / 1 hr.	0.26% / 4.8 hrs.	0.40% / 14 hrs.
(d)	0% /13 secs.	0% /1.5 hrs.	0.01% / 3.5 hrs.	0.06% / 13 hrs.

Fig. 3. Results from the A-Teams of Fig. 2 for four representative TSP problems. The results are averages over 15 runs. Each run was terminated when improvements in the tours ceased. All the agents of each A-Team were made to share a single computer - a DEC 5000. Lower times were obtained when more computers were provided with the larger A-Teams benefiting more than the smaller ones. For instance, with 4 computers for the ATT 532 problem, the time for (a) decreased from 7.5 to 5 hours while the time for (d) decreased from 13 to 3.4 hours.

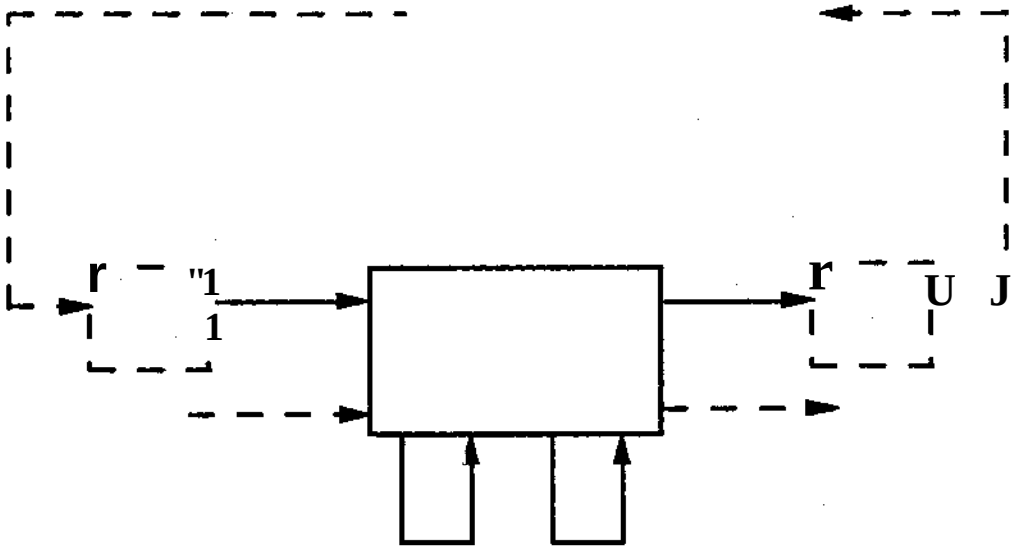

Fig. 4. From a memory's viewpoint, the rest of an organization is a set of agents.

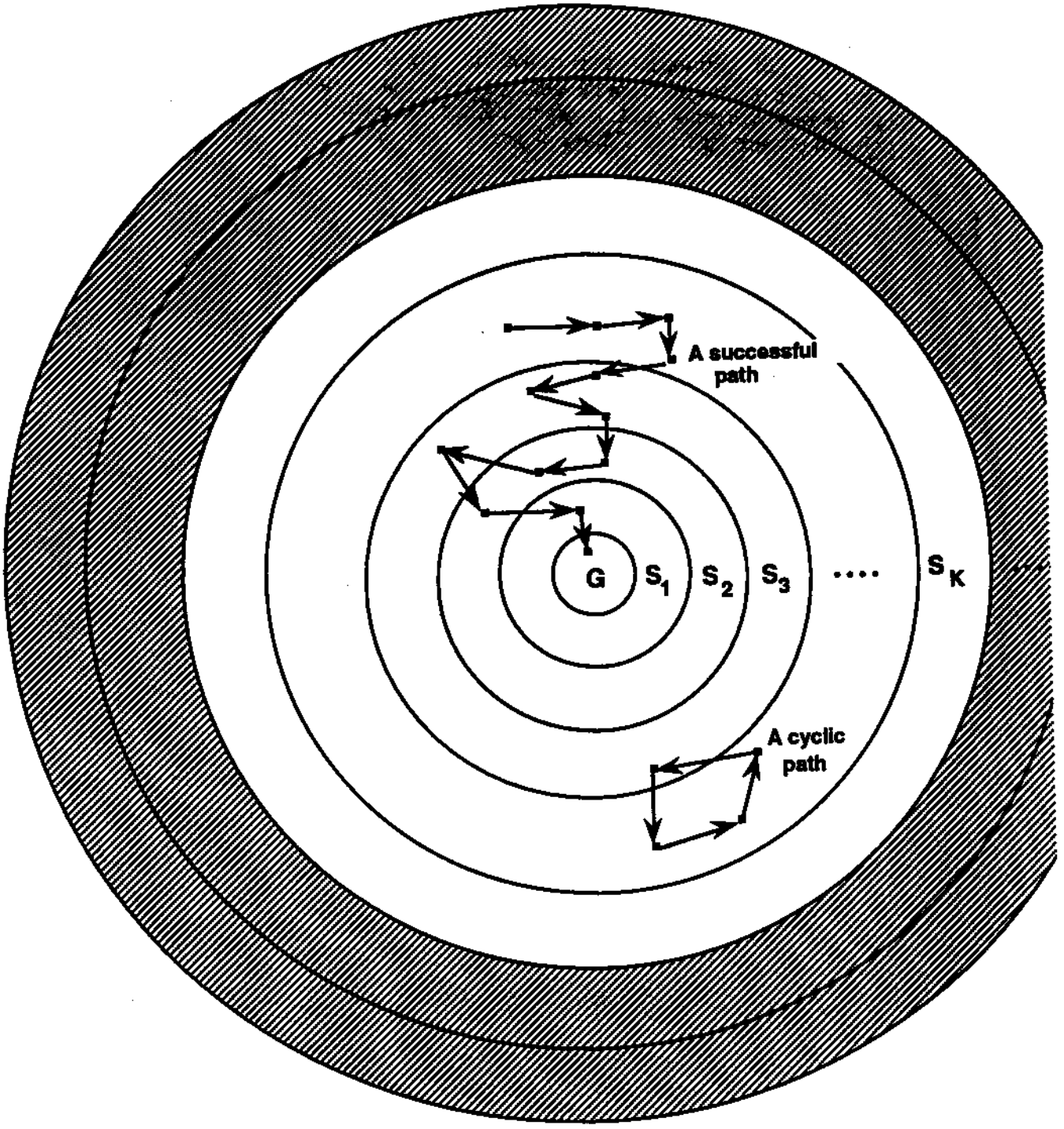
A successful path
G
S_1
S_2
S_3
....
S_K
A cyclic path

Fig. 5. The space of solutions is partitioned into regions so that all the points in Sn are n-constructive operations from the goal space, G. This partition depends on C, the contingent of constructive agents. As C is expanded, the space shrinks about G in the sense that some points that were far from G move closer.