ASYNCHRONOUS TEAMS: COOPERATION SCHEMES FOR AUTONOMOUS AGENTS

Sarosh Talukdar
Lars Baerentzen
Andrew Gove
Pedro de Souza

Carnegie Mellon University
Pittsburgh, PA 15213

Contact Person: Sarosh Talukdar
ECE, Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213.
Tel: 412-268-8778
E-mail: talukdar@cmu.edu

Copyright 1996 by Talukdar

ABSTRACT
Experiments over a variety of optimization problems indicate that scale-effective convergence is an emergent behavior of certain computer-based agents, provided these agents are organized into an asynchronous team (A-Team). An A-Team is a problem-solving architecture in which the agents are autonomous and cooperate by modifying one another's trial-solutions. These solutions circulate continually. Convergence is said to occur if and when a persistent solution appears. Convergence is said to be scale-effective if the quality of the persistent solution increases with the number of agents, and the speed of its appearance increases with the number of computers. This paper uses a traveling salesman problem to illustrate scale-effective behavior and develops Markov models that explain its occurrence in A-Teams, particularly, how autonomous agents, without strategic planning or centralized coordination, can converge to solutions of arbitrarily high quality. The models also predict two properties that remain to be experimentally confirmed:

- construction and destruction are dual processes. In other words, adept destruction can compensate for inept construction in an A-Team, and vice-versa. (Construction refers to the process of creating or changing solutions, destruction, to the process of erasing solutions.)
- solution-quality is independent of agent-phylum. In other words, A-Teams provide an organizational framework in which humans and autonomous mechanical agents can cooperate effectively.

1. INTRODUCTION

All the available algorithms for optimization and constraint satisfaction have weaknesses--the rigorous algorithms tend to be too slow, the heuristics, too unreliable. Rather than trying to design a new algorithm without weaknesses, a task that is difficult if not impossible, we have been working on ways to organize algorithms so they can suppress their weaknesses through cooperation, and together do what separately they might not. The result is a type of organization, called an asynchronous team (A-Team), that combines features from a number of systems, particularly, insect societies [1], cellular communities [2], genetic algorithms [3], blackboards [4], simulated annealing [5], tabu search [6], and brainstorming [28].

1.1 What is an A-Team?

> Definition. An A-Team is a set of autonomous agents and a set of memories, interconnected to form a strongly cyclic network, that is, a network in which every agent is in a closed loop.

An A-Team can be visualized as a directed hypergraph, called a data flow, like those in Fig. 1. Each node represents a complex of overlapping memories. Each arc represents an autonomous agent. Results or trial-solutions accumulate in the memories (just as they do in blackboards) to form populations (like those in genetic algorithms). These populations are time varying: new members are continually added by construction agents, while older members are being erased by destruction agents. More specifically, a construction agent copies results from the population at its tail, modifies the copies and inserts them into the population at its head. A destruction agent examines a population and erases those results it feels should not be there, often working from a list of results to be avoided (like the lists used in tabu searches).

The numbers of construction and destruction agents can be arbitrarily large and each agent, whatever its type, can be arbitrarily complex. Consequently, the problem-solving skills of a data flow can be arbitrarily apportioned between construction and destruction. (Other synthetic problem-solving systems invariably concentrate on one or the other. Hill climbing, for instance, concentrates on how to construct new and better solutions while simulated annealing, genetic algorithms and tabu search concentrate on how to destroy or reject weak solutions. Natural systems, however, often benefit from a more symmetric use of construction and destruction. 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.)

All the agents in an A-Team are autonomous. An autonomous agent decides for itself what it is going to do and when it is going to do it (like the adult members of insect societies). Consequently, there can be no centralized control. But new autonomous agents can be easily added (there is no supervisory hierarchy to get in their way).

Agents cooperate by working on one another's results. Because the agents are autono-

mous, this cooperation is asynchronous (no agent can be forced to wait for results from another). Rather, all the agents can, if they so choose, work in parallel all the time. (Other synthetic problem-solving systems often include precedence constraints to force at least a partial order on the activities of their computational modules. Traditional genetic algorithms, for instance, require destruction to cease while construction is in progress, and vice-versa.)

1.2 The Effects of Scale

One might think that A-Teams would be prone to anarchy. After all, every agent does what it wants when it wants, and makes its decisions without knowing anything about the other agents except for the results they produce. Nevertheless, useful A-Teams have been developed for a wide variety of optimization and constraint satisfaction problems, including, nonlinear equation solving [7], [24], traveling salesman problems [14], high-rise building design [8], reconfigurable robot design [9], diagnosis of faults in electric networks [10], control of electric networks [11], [25], job-shop-scheduling [16], steel and paper mill scheduling [26], [27], train-scheduling [15], and constraint satisfaction [18]. Not only do these A-Teams produce very good solutions, but they appear to be scale-effective.

> Definition. An organization is scale-effective if its performance improves with size. A computer-based organization is scale-effective if there are agents whose addition improves solution-quality and computers whose addition improves solution-speed.

Scale-effectiveness is an extremely desirable property. The problem of improving the performance of a scale-effective organization reduces to one of finding which components to add. A non-scale-effective organization faces the much more difficult problem of finding which of its parts to eliminate or modify before additions can be of benefit. Synthetic organizations are often non-scale-effective, being described by the proverb, “too many cooks spoil the broth.” That is, there comes a point when the addition of another “cook,” no matter how competent she may be, has a negative impact on overall performance. In a scale effective organization there can never be too many “cooks.”

Scale-effective behavior will be illustrated later in the paper by the application of A-Teams to the TSP (traveling salesman problem: given M cities and their separations, find the shortest tour or closed path that goes through all the cities).

1.3 About The Rest Of The Paper

The remainder of the paper is organized as follows. Section 2 covers the structure of A-Teams and presents a representative sampling of experimental results. Section 3 develops a Markov model that explains these experimental results and predicts some behaviors that remain to be experimentally confirmed. Section 4 summarizes the most important parts of the preceding sections. Finally, the appendix provides the mathematical details of the Markov model.

2. A DESIGN SPACE FOR A-TEAMS

Let {Df} be a design space for A-Teams, that is, a set of A-Teams represented in terms of their design variables. More specifically, let {Df} be the set of every possible hypergraph (data flow): Df = (V, W), where V is a set of nodes (memories) and W is a set of arcs (agents).

The process of designing an A-Team to solve a given optimization problem, X, is equivalent to searching {Df} for a data flow Df*, such that a solution to X of appropriately high quality will appear in one of the memories of Df* in an appropriately short time after Df* has been activated.There is no automatic procedure for conducting this search, nor is one likely to become available in the immediate future. Instead, as with many other design tasks, the best that can be done is to decompose the overall design space into more manageable subspaces. A decomposition that we have found to be helpful is given below. The searches through the subspaces are labeled as steps and illustrated with some A-Teams we built for the TSP (traveling salesman problem).

Step 1. Choose a superproblem.

Definition. A superproblem is a set of problems containing X and problems related to X, such as parts of X, relaxations of X, and other problems whose solutions might provide clues to good solutions of X.

When X is the TSP in M cities, three related problems are: a) find good tours of the M cities, b) find good partial tours of the M cities, and c) find good 1-trees of the M cities; where "good" means "containing many of the arcs of an optimal tour" and a 1-tree is a graph with M arcs that connects all the cities but does not always form a tour (the 1-trees are a superset of the tours and hence, the shortest 1-tree provides a lower bound on the shortest tour).

Step 2. Choose the node-set, V, by assigning one or more memories to each member of the superproblem.

The purpose of each memory is to hold a population of trial-solutions to its problem. As in the case of genetic algorithms, larger populations lead to better solutions but require greater computing efforts. However, the marginal benefits to solution-quality fall off rapidly. In other words, moderately sized populations produce solutions that are almost as good as those from large populations.

After some experimentation, we choose the population size for the TSP to be the same as M, the number of cities (M/2 and 2M seem to work almost as well as far as solution-quality is concerned). We used ordered lists of cities as the representation for complete and partial tours. With this representation, {Atlanta, Boston, Raleigh, Pittsburgh} means a tour that goes from Atlanta to Boston to Raleigh to Pittsburgh, and then back to Atlanta. The representation chosen for 1-trees is more complicated. Interested readers can find it in [14].

Step 3. Choose a set of construction algorithms or operators for the members of the superproblem.

In our experience, the greater the range of skills of these algorithms, the better the solu-

tions that will be found. The algorithms do not have to be uniform in size or coverage. Rather, some can be large, others small, some can be general, others specialized. Weak but fast algorithms, such as crossover and mutation, when used exclusively, or powerful but slow algorithms, such as branch-and-cut, when used alone, will invariably do less well than a mix that includes both.

Rigorous algorithms that can solve large TSPs in reasonable (polynomial) amounts of time are unknown. However, there are dozens of faster heuristics, several of which are available as computer codes. One of the better heuristics is the Lin-Kernighan (LK) algorithm [12]. A variety of simpler heuristics is listed in Fig. 2.

Step 4. Form each construction algorithm into an autonomous agent.
Think of an agent as operating between two worlds--one it perceives, the other it affects. These worlds may overlap and can be thought of as input and output memories (repositories for the objects the agent is able to perceive and affect).

> Definition. An agent consists of an input-memory, an output memory, an operator and a control system.The operator modifies objects obtained from the input-memory and places the results in the output-memory. The control system decides which objects will be obtained and when, that is, the control system consists of a selector (to choose objects from the input-memory for the operator to work on) and a scheduler (to determine when the operator will work, and which of the tools and other resources available to the agent will be used in doing this work).

This definition of agency is broad enough to include everything from a simple program with some input-selectivity, through intelligent robots, to humans.

> Definition. An agent is autonomous if its control system is completely self-contained, that is, if it accepts no selection or scheduling instructions from other agents.

One might expect that effective cooperation among autonomous agents would be contingent on the use of good selection strategies. In our experience, “good” does not have to mean “complex.” When solution-quality is measured by a single attribute (such as tour length), a very simple selection strategy seems to work quite well. This strategy is: select solutions randomly with a bias towards the better solutions. Murthy [9] describes a variant for cases where solution-quality is best measured by a vector of conflicting attributes. Specifically: compare a vector representing the estimated needs of the solution to a vector representing the estimated capabilities of the agent; then arrange for the probability of selection to increase as the magnitude of the angle between these two vectors decreases.

While scheduling strategies are undoubtedly important, we have yet to investigate their effects. For the TSP, and all the other cases we have studied, we have used only one very simple strategy: allow each agent to run continuously, or as close to continuously as the available computers will permit.

Step 5. Choose autonomous destroyers.
The construction agents produced in the previous step are designed to add trial-solutions to their output-memories. Their actions must be balanced by destroyers--agents that erase trial-solutions--or the memories would soon overflow. There are two additional functions that destroyers can perform. First, they can improve the overall quality of solution-populations by erasing their weakest members. Second, they can terminate undesirable patterns of construction, such as repeating sequences of solutions, by recognizing and erasing them as soon as they occur.

Since the operation performed by a destroyer--erasure--requires no skill, all the intelligence of a destroyer is concentrated in its selector and scheduler. There are three sources of selection technology for erasing or rejecting solutions: genetic algorithms, simulated annealing and tabu search. The first two of these use essentially the same process: random selection biased by a scalar merit function to make solutions with greater merit less likely to be selected for destruction. We chose this process for the TSP. Specifically, tours and 1-trees were selected randomly for erasure, with the longer tours and 1-trees being more likely to be selected than the shorter ones. The destroyers were scheduled to act so there were always a few slots left open for the constructors to fill.

Step 6. Choose the arc-set, W, that is, choose an interconnection of agents and memories that is strongly cyclic.
Implement the resulting data flow in a network of computers. Seed the memories with initial populations of trial-solutions, activate the agents and monitor the resulting population dynamics. If convergence is overly slow, that is, if at least one complete solution of acceptable quality to X, the problem to be solved, is overly slow in appearing, then add more cycles to the data flow, either by repeating from step-1or from step-3. The assumption in both cases is that the data flow is scale-effective, and therefore, there are always cycles whose addition will produce better convergence.

The process of designing data flows by “repeating from step-1” is illustrated in Fig. 1. The first iteration, Fig. 1 (a), contains a single memory for tours. The algorithms include LK, the most powerful tour-improvement algorithm we were able to acquire, CLK, a simpler and faster version of LK, as well as OR and D1, which are much simpler algorithms chosen to compensate for LK’s and CLK’s weaknesses. (LK and CLK are not iterative algorithms. When either is applied to a population of tours, all the improvements it can make are obtained after its first application to each tour. For difficult TSPs, it is likely that this application will leave some of the tours unchanged, and none improved to optimality. The purpose of OR was to make possible repeating patterns in which a tour that was improved by LK or CLK could be further improved by OR, and in the process, changed enough so it could again be improved by LK or CLK. The purpose of D1, a random destroyer, was to keep the population in check.) The resulting data flow was able to find optimal solutions far more frequently than LK [14]. The other data flows in Fig. 1 were obtained by adding more complex cycles, each intended to produce more tours that LK or CLK could improve. Notice the effects of scale on solution-quality, especially, how quality increases with the size of the data flow (Fig. 3).

The process of designing data flows by "repeating from step-3" is, in our experience, a little easier, perhaps because the designer can concentrate on the strengths of the agents instead of their weaknesses. The node-set is fixed during the first and only pass through steps 1-2, and is made as comprehensive (large and diverse) as practical. In the iterations through steps 3-6, the designer seeks to increase the diversity of the arc-set (the range of skills of the algorithms), until a suitably wide range is obtained. The simpler algorithms can be added first allowing the more complex and usually more troublesome algorithms to be added later, if and when their skills are needed. For example, consider the node-set of Fig. 1(d). Agents can be added in almost any order to provide improving solutions. And in some orders that go from simple to complex, there is also a progressive improvement in solution-speed, even though all the agents must share a single computer (Fig. 4). With more computers, all the data flows, but especially the larger ones, can be further speeded-up (e.g. Fig. 5).

It must be emphasized that the A-Teams of Fig. 1 are not intended to be competitive with the leading TSP procedures. Rigorous algorithms (e.g. [31]) always find optimal tours, if given enough time, and the leading heuristics [29], [30], [32]-[34] are far faster. Rather, the purpose of Figs. 1-5 is to illustrate how distinct algorithms can be combined into teams and what benefits these teams can provide.

3. COOPERATION IN STRONGLY CYCLIC DATA FLOWS

Why do A-Teams behave as they do? What are the underlying phenomena and causal relations? We will tackle these questions with the aid of a model called a CDM (constant drift memory). The justification for the use of this model, in outline, is as follows:

- all forms of cooperation can be represented by data flows;
- in any data flow, all the behaviors of interest occur in only one memory;
- this, and all the other memories in any strongly cyclic data flow, can be accurately modeled by devices called cyclic memories;
- all cyclic memories can be modeled by a certain simple type of cyclic memory, namely, a CDM.

The technical apparatus needed to make this justification is given in some detail below and greater detail in the Appendix.

3.1 Data Flows As Universal Models Of Cooperation

> Definition. Two agents cooperate when the input-space of one has a non-zero intersection with the output-space of the other, that is, when the agents can exchange results, regardless of whether the exchanges are productive or not.

By this definition, all the ways in which agents can cooperate are representable by data flows. We believe this definition includes all possible interactions among problem-solving agents. This being so, all forms of cooperation among problem-solving agents can be represented by data flows.

Recall that each memory in a data flow is dedicated to holding a population of trial-solutions to one member of a superproblem. Obviously, the dynamics of these populations are determined by their initial values and by the agents that act on them. We will examine

these dynamics in what follows.

3.2 Terminology

Definition. A memory is a cyclic memory if all the agents that write to it also read from it.

Thus, any memory in a strongly cyclic data flow can be modeled by a cyclic memory. After all, a memory only has direct contact with those agents that read from or write to it. If the data flow is strongly cyclic, then most, if not all, the agents that do one of these things can also be thought of as doing the other. For instance, the entire subgraph that begins with agent-AI and ends with agent Dec in Fig 1 (d), can, from the perspective of the partial tour memory, be replaced by a single super-agent that both reads from, and writes to, the partial tour memory.

Definition. A cyclic memory is a constant drift memory (CDM) if:

- The members of its initial population of solutions are chosen randomly from the space of all possible solutions that can be stored in the memory. And the size of this population is small in comparison to the size of the space of all possible solutions.
- A path (a connected sequence of solutions) is developed from each of the members of the initial population by the combined actions of constructors and destroyers. A constructor, when it chooses to act on a path, lengthens it by adding a point to its end; this point is a modification of its immediate predecessor. A destroyer, when it chooses to act on a path, shortens it by removing a point from its end. The aggregate results of the actions of the constructors and destroyers are described by a Markov chain of the sort shown in Fig. 6.
- All the paths are developed concurrently. The total time required for the development of each path is the sum of the time that agents spend actually working on the path (adding or erasing points) plus the time by which they are delayed in their work. These delays are of three types: synchronization delays that occur when an agent must pause in order to satisfy a synchronization or precedence constraint in the organization's control structure, communication delays that occur when an agent must wait for the delivery of the data it needs, and resource contention delays that occur when an agent must wait for the computers it requires.
- The memory and its agents are implemented in a network of computers. Each agent is assigned a computer for its exclusive use, so there are no resource contention delays. Moreover, this computer is sized so that each agent requires the same amount of time for an action as every other agent (that is, the power of the computer assigned to each agent is proportional to the agent's computational needs).

Consider any CDM. Let:

C be the set of construction agents that acts on the CDM.

θ_C be the set of operators contained in C.

D be the set of destruction agents that acts on the CDM.

S be the space (set) of all possible solutions, good and bad, that can be stored in the CDM.

H(S) be the set of all the paths in S, a path being a sequence of one or more points. Therefore, $S \subset H(S)$.

$H_D(S)$ be the subset of H(S) that contains those undesirable paths (such as points that are very far from G_δ, overly long paths and orbits) that the destroyers can recognize and erase.

Definition. A point, s, is inaccessible if $s \in H_D(S)$, that is, if the destroyers in D recognize and erase the point before it can be selected by a construction agent. Thus, construction agents are prevented from developing paths that begin at inaccessible points.

Consider any CDM. Let:

δ be an indicator of solution-quality, such that δ increases with solution-quality.

G_δ be the subset of S that contains all the solutions of quality δ and better.

N be the size of the initial population of solutions stored in the CDM. Assume that the members of this population are chosen randomly from S and that N is always small in comparison with the size of S.

T_δ be the expected amount of time for the population of solutions to evolve at least one solution of quality δ or better.

Definition. G_δ is reachable if T_δ is finite. Moreover, the performance of a CDM is the double: (δm, vm), where δm is the highest quality solution that will appear in the memory, that is, δm is the greatest value of δ such that G_δ is reachable, and vm is the expected speed with which this solution appears, that is, $vm = 1/T_{\delta m}$ where $T_{\delta m}$ is the expected time for reaching $G_{\delta m}$.

Consider any CDM. Let:

$d_C(y)$ be the distance of y from G_δ, where y is any solution in S, and $d_C(y)$ is the minimum number of successive actions by construction agents from C that are needed to convert y into a member of G_δ. Thus, $d_{C2}(y) \leq d_{C1}(y)$ when $C1 \subseteq C2$.

P(S, C, D, δ) be a partition of S into regions S_0, S_1,..., S_∞, and I, such that $S_0 = G_\delta$; S_n contains all the solutions that are at a distance of n from S_0; and I contains all the points that are made inaccessible by the destroyers in D; as in Fig. 7.

μ be the mean distance of the accessible points to G_δ.

L_{exp} be the expected length of the path from the best point in the initial population of solutions to G_δ (it is assumed that this point is accessible).

β be the amount of time required for each agent to take one action.

T_{syn} and T_{com} be the expected synchronization and communication delays experienced by agents in developing a successful path (one that reaches $G_{\delta m}$).

Consider s_i and s_{i+1}, the two latest points in any developing path. Assume that both points are accessible and neither is in G_δ.Let:

p, q and r be the probabilities that s_{i+1} is closer, further and at the same distance, respectively, as s_i from G_δ.

p_c, q_c and r_c be the values of p,q and r when the destroyers are disabled.

p_d, q_d and r_d be the conditional probabilities that s_{i+1}will be destroyed, if it is considered for destruction and if it is further, closer and at the same distance, respectively, as s_i from G_δ.

$\lambda(s_i) = p - q$ be the overall drift of the CDM at s_i; $\lambda_c(s_i) = p_c - q_c$, be the component of drift contributed by the constructors; and $\lambda_d(s_i) = p_d - q_d$ be the component of drift contributed by the destroyers. (Note: in a CDM, λ, λ_c and λ_d are constants for all accessible points at finite and non-zero distances from the goal. But in other cyclic memories they could vary. Also, it is possible for $p_d + q_d + r_d$ to be different from 1.)

$\Lambda(S)$ be the space of all the $\lambda(s_i)$.

Consider P(S, C, D, δ). Its configuration can be of several different types. Three are of interest here:

P1 = {P(S, C, D, δ) | S_∞ is empty or inaccessible},

P2 = {P(S, C, D, δ) | S_∞ is non-empty and accessible}, and

P3 = {P(S, C, D, δ) | S_∞ is non-empty, accessible and includes all of S not in G_δ}.

3.3 Reachability Conditions And Causal Relations

For any CDM:

- if λ is positive and P(S, C, D, δ) $\in$ P1, then G_δ is reachable;
- the causal relations among the variables are as depicted in Fig. 8.

The proof is given in the Appendix.

3.4 CDM Behavior

What do the reachability conditions and causal relations mean?

3.4.1 Reachability

Paraphrased, the reachability conditions for a CDM say that if the mean distance (μ) of the accessible points to the goal (G_δ) is finite, and if the drift (λ) is positive, then the expected length (L_{exp}) of paths developed from accessible points to the goal by the combined actions of the constructors and destroyers, will also be finite.

The partition P(S, C, D, δ) makes two facts obvious:

- μ is finite if and only if S_∞ is empty or inaccessible; and
- μ increases as δ increases, that is, as G_δ shrinks.

Less obvious but equally true: μ decreases with certain expansions of C or D. To understand why, note that by definition, $d_C(y)$, the distance from any point y in S to G_δ, cannot increase as C expands and will decrease with certain expansions. In other words, expanding C tends to cause a migration of solutions from the outer regions of P(S, C, D, δ) to its inner regions. For instance, consider the problem of finding optimal tours for the TSP. If the set of construction agents is expanded by the addition of LK, then all the solutions that LK can improve to optimality will move from their positions, however distant from the goal, into S_1, the region that is just one action away from the goal. As another

illustration, when LK is used alone, all the solutions that it cannot improve to optimality are in S_∞. But when any other construction agent is added, all the solutions that can be iteratively improved to optimality by the alternate application of LK and this new agent, are moved out of S_∞ into regions closer to the goal.

Certain expansions of D have a similarly beneficial effect. In particular, additions of destroyers that make the outermost regions of P(S, C, D, δ) inaccessible will reduce μ. For instance, if LK is the sole construction operator, then adding destroyers that can recognize and quickly erase all solutions that LK cannot improve to optimality, would cause all the accessible, non-optimal solutions to be in S_1, and μ to decrease from ∞ to 1.

Thus, finding solutions of arbitrarily high quality in a CDM requires neither strategic planning nor coordination. Rather, agents acting independently, without central control, can find these solutions provided only that the skills of the agents, in aggregate, are sufficiently large and diverse to make μ finite and λ positive. μ is decreased by the addition of construction agents whose operators have new skills and by the addition of destruction agents that make distant solutions inaccessible. The value of λ depends on the values of λ_c and λ_d, the drifts of the individual constructors and destroyers. Recall that these individual drifts are measures of selection acuity. A constructor has a positive value for λ_c when the solutions it selects to work on, are moved closer to the goal more often than further away. A destroyer has a positive value of λ_d when its decisions to erase solutions are correct more often than wrong.

Expressions for the dependence of λ on λ_c and λ_d can be found in the Appendix and visualized with the aid of curves of the sort shown in Fig. 9. This dependence is such that destruction has little effect on overall drift when the construction agents make relatively few selection errors ($\lambda_c > 0.2$). But when construction agents are prone to making numerous selection errors, high overall drifts can still be obtained by using destroyers to prevent these errors or repair their effects, the former, by making inaccessible the points that construction agents shouldn't select, the latter, by erasing erroneous construction patterns, such as orbits and paths heading in the wrong direction. As a simple illustration, an LK-based construction agent could be kept from making any selection errors by destroyers that erase all solutions that LK cannot improve to optimality, as soon as these unimprovable solutions appear.

3.4.2. Solution-Speed

The causal diagram of Fig. 8 indicates that the addition of agents (expansions of C or D) will improve solution-speed if the completely solid paths from C and D to quality and speed dominate the paths containing broken arcs (because the solid paths represent relationships that are known to be monotonically improving while the broken arcs represent indeterminate relationships).

In an organization with a centralized control system, agents suffer synchronization delays whenever they must pause in their work to satisfy the precedence constraints that such systems invariably impose. These synchronization delays can be significant. But

autonomous agents respect no precedence constraints and therefore, have no synchronization delays. Thus, the exclusive use of autonomous agents causes the T_{syn} node to disappear from the causal diagram, along with all the broken paths that go through it. Consequently, the conditions under which solution-quality and speed become commensurate (both increasing as agents are added) are easier to meet. So much so, that quality and speed may both improve even as the average amount of computer power per agent is decreasing. Fig. 4 is a case in point. Recall that a CDM requires each agent to be assigned its own computer, sized so that an action by the agent takes the same amount of time as an action by any other agent. Thus, when an agent that has been working continuously completes n actions, all the other agents that have been working continuously will also have completed roughly the same number of actions. This situation is simulated by the "round-robin" scheme of Fig. 4: no agent is allowed to begin its (n+1)-th action until all the other agents have completed their n-th actions. Of course, since all the agents must share a single computer, the time to complete each round of actions increases with the number of agents. But in the case of Fig. 4, the number of "rounds" needed seems to decrease faster than the rise in time per "round," so overall speed increases as new agents are added.

3.4.3. Population Size

The causal diagram (Fig. 8) indicates that, N, the size of the solution-population, has no affect on solution-quality. The explanation is in 3.4.1. Briefly, solution-quality (δm) is determined solely by when μ becomes infinite and λ becomes negative, events that depend only on the structure of $P(S, C, D, \delta)$ and the selection acuity of the agents.

The causal diagram does indicate improvements in solution-speed with increases in N, but an examination of the details in the Appendix reveals that these improvements are prominent only in the early stages of path-development and are prone to saturation.

3.4.4. The Mechanics of Expansion

How difficult is it to add an agent to a CDM? The reachability conditions place no restrictions on agent-type or granularity. Therefore, as far as solution-quality is concerned, large agents can be mixed with small agents, general agents with specialists, human agents with computer-based agents, and autonomous agents with non-autonomous agents.

If an organization contains non-autonomous agents, then it must provide the supervision these agents need. And the addition of a new type of non-autonomous agent may necessitate expensive modifications to the supervisory system. However, if each agent comes with its own complete and self-contained control system, as is the case with autonomous agents, then no modifications are necessary. Furthermore, the control system can be customized for the agent's operator: large, complex operators can be paired with appropriately large and complex controls, small operators, with simple controls.

3.4.5. Duality

In a CDM, constructors and destroyers are similar in their effects on solution-quality and speed. In other words, adept destruction can compensate for inept construction, and

vice-versa. The argument is as follows.

First, the addition of both destroyers and constructors reduces μ, the mean distance to the goal. Destroyers do this by making distant solutions inaccessible, constructors, by moving solutions closer to the goal.

Second, the sensitivity of the overall drift, λ, to the constructor drift, λ_c, is similar to its sensitivity to destroyer drift, λ_d (Fig 9).

Since μ and λ are the sole determinants of reachability, it follows that constructors and destroyers are similar, if not equivalent, in their effects on solution-quality. This is confirmed by the near-symmetry of the subgraphs connecting C and D to δm in the causal diagram (Fig. 8). (The symmetry means that the relations between C, the set of constructors, and δm, the highest quality of solution that is reachable, are similar to the relations between, D, the set of destroyers, and δm.) Notice also that the subgraphs connecting C and D to vm are symmetrical indicating that constructors and destroyers have similar, if not equivalent relationships with solution-speed.

3.5 Design Rules For A-Teams

There are structural differences between A-Team memories and CDMs. Specifically, in an A-Team memory: a) an agent may use several old solutions, rather than just one, to produce a new solution, b) the overall drift or rate of diffusion of accessible solutions towards the goal is likely to vary, rather than be constant, and c) there may be many fewer computers than agents. We believe these differences to be inconsequential to the conclusions reached in section 3.4. Indeed, empirical results [7-11, 14-18, 24-27] confirm the conclusions on reachability, population-size, and the mechanics of expansion for A-Teams built exclusively from computer-based agents. Duality, however, remains to be experimentally confirmed, as does the prediction that humans can as easily and effectively be incorporated into A-Teams as computer-based agents.

With regard to solution-speed, we believe that CDMs provide optimistic estimates that will be approached by A-Teams as the numbers of computers are increased till every agent has its own, dedicated computer.

All this being so, the conclusions of section 3.4 can be distilled into the following rules: If memories and autonomous agents are connected into a strongly cyclic network (so there are no synchronization delays, results can circulate freely and agents can cooperate by modifying one another's results), then

- the network tends to be scale-effective; solution-quality tends to increase with the number of construction agents and particularly, with the diversity of their operators; solution-speed tends to increase with the number of computers till every agent has its own computer.
- construction and destruction are dual processes; adept destruction can compensate for inept construction, and vice-versa. (A corollary is that knowledge can and should be placed where it fits best. Specifically, knowledge on what to do should be put into construction agents, knowledge on what to undo, into destruction agents).

- performance is independent of agent granularity and type. Therefore, designers should feel free to mix large agents with small ones, and mechanical agents with humans.

4. SUMMARY AND CONCLUDING REMARKS

The function of an A-Team is to combine operators or algorithms, so together they can tackle bigger and more difficult problems than they could if working alone. A-Teams have been shown to be useful for problems whose algorithm sets are rich but imperfect, that is, problems for which many different sorts of algorithms are available but no single algorithm is entirely satisfactory.

Structurally, an A-Team is a strongly cyclic network of memories and autonomous agents. Each memory is dedicated to one problem. Collectively, the memories represent a superproblem or cover of the problem-to-be-solved. Trial-solutions to members of the superproblem are produced by the agents and stored in the memories to form populations. Agents cooperate by working on one another's solutions, and are divided into two types by the sort of work they do. Specifically, construction agents add solutions to populations and destruction agents erase solutions from populations.

Agents are defined broadly enough to include all manner of problem-solving entities, including computer-based agents and humans. Specifically, an agent is defined to consist of three components: an operator (algorithm), a selector and a scheduler. The operator creates or modifies trial-solutions, the selector determines which solutions the operator will work on, and the scheduler determines when this work will be done and with what resources. In other words, the skills of the agent to create or modify solutions are resident in its operator, the intelligence with which it applies these skills is resident in its selector and scheduler. An agent is autonomous if its selector and scheduler are completely self-contained.

The networks that constitute A-Teams can be visualized as directed hypergraphs, called data flows. We have found it helpful to decompose the task of designing such networks (searching the space of all data flows) into six subtasks: design a superproblem, design memories to store populations of trial-solutions to members of the superproblem, select a set of construction algorithms, combine these algorithms with selectors and schedulers to form autonomous construction agents, design a set of autonomous destruction agents, and form the constructors, destroyers and memories into a strongly cyclic network.

While none of the structural features of an A-Team is unique, their combination is unusual, and the behaviors of A-Teams can be counter-intuitive. How are these behaviors to be understood? There is reason to believe that the memories of A-Teams and certain analytical devices, called CDMs (constant drift memories), have the same causal mechanisms. This being so, CDMs can be used not only to understand A-Teams but also to help design them. Specifically, CDMs provide reachability conditions (section 3.3) and causal relations (Fig. 8) by which to improve solution-quality and speed, when such improvements are needed.

Besides CDMs, three useful conceptual aids are: P(S, C, D, δ), a partition of all possible trial-solutions by their accessibility and distance from the goal; $H_D(S)$, a partition of all possible construction patterns by their desirability; and $\Lambda(S)$, a field of the net rates at which solutions will drift or diffuse towards the goal. P(S, C, D, δ) provides a view of the solutions that are accessible for modification by the construction operators, and what the these operators could do, if they were controlled perfectly; $H_D(S)$ provides a view of the mistakes the construction-operators can make, if they are controlled imperfectly; and $\Lambda(S)$ shows how well the constructors and destroyers will actually do when they are working together.

The CDM reachability conditions indicate that the task of obtaining solutions of acceptable quality in any single memory can be broken into two sub-tasks: a) make the outer regions of P(S, C, D, δ) empty or inaccessible, and b) make $\lambda(s)$, the net drift at solution s, positive and as large as possible, for all accessible values of s. Any mix of construction and destruction agents that accomplishes these sub-tasks will, if the agents are allowed to work on one another's trial-solutions, ensure that some of these solutions will eventually achieve an acceptable level of quality. Moreover, if the agents are autonomous and free of resource constraints, they can work in parallel all the time, and there is good reason to believe that solution-speed improves if they do.

In more practical terms, these conclusions can be stated as the following design rules: If memories and autonomous agents are connected into a strongly cyclic network, then
- the network tends to be scale-effective, so adding agents tends to improve solution-quality and adding computers tends to improve solution-speed.
- construction and destruction are dual processes, so adept destruction can compensate for inept construction, and vice-versa.
- performance is independent of agent granularity and type, so large agents can be mixed with small ones, and mechanical agents with human beings.

Parts of these rules have been confirmed by experiment, at least for optimization problems. The remainder--construction as the dual of destruction and the possibility of effective cooperation between humans and autonomous mechanical agents--are predictions from the analysis of CDMs that are still to be experimentally verified.

The rules say nothing about two important steps in the design of an A-Team: how is the superproblem to be selected? and, what scheduling strategies should the agents employ? So far, these steps have been performed by trial-and-error. Models other than CDMs will be required to develop more systematic procedures.

REFERENCES

[1] 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, Springfield, 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.
[11] S. N. Talukdar, V.C. Ramesh, "A Multi-Agent Technique for Contingency Constrained Optimal Power Flows," *Proceedings of the Power Industry Computer Applications Conference* (PICA-93), Phoenix, May, 1993,*IEEE Trans. on Power Systems,* Vol. 9, No. 2, May 1994, pp. 855-861.
[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 Department, CMU, Pittsburgh, PA, 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] J. A. Lukin, A. P. Gove, S. N. Talukdar and C. Ho, "Automated Probabilistic Method for Assigning Backbone Resonances of (^{13}C, ^{15}N)-Labelled Proteins," Journal of Biomolecular NMR, 9, 1997.
[18] S. R. Gorti, S Humair, R. D. Sriram, S. Talukdar, S. Murthy, "Solving Constraint Satisfaction Problems using ATeams," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* Vol. 10, pp. 1-19, 1996.
[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 and 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, “Polyedrische Kombinatorik and Schnittebenverfahren,” Preprint No. 38, Universitat Augsburg, 1984.
[23] M. Padberg and G. Rinald, “Optimization of a 532-city Symmetric Traveling Salesman Problem,” *Operations Research Letters*, Vol. 6, No. 1, March 1987.
[24] S. N. Talukdar, S. S. Pyo and T. Giras, “Asynchronous Procedures for Parallel Processing,” IEEE Trans. on PAS, Vol. PAS-102, NO 11, Nov. 1983.
[25] P. Avila-Abascal and S. N. Talukdar, “Cooperative Algorithms and Abductive Causal Networks for the Automatic Generation of Intelligent Substation Alarm Processors”, Proceedings of ISCAS-96.
[26] J. Rachlin, F. Wu, S. Murthy, S. Talukdar, M. Sturzenbecker, R. Akkiraju, R. Fuhrer, A. Aggarwal, J. Yeh, R. Henry and R. Jayaraman, “Forest View: A System For Integrated Scheduling In Complex Manufacturing Domains,” IBM report, 1996.
[27] H. Lee, S. Murthy, W. Haider, D. Morse, “Primary Production Scheduling at Steel making Industries,” IBM report, 1995
[28] S. Pugh, “Total Design,” Addison Wesley, 1990.
[29] D. S. Johnson, “Local Optimization and the Traveling Salesman Problem,” Lectures in Computer Science, Automata, Languages in Programming, 17-th Int. Coll., Springer-Verlag, Vol. 443, July, 1990
[30] O. Martin, S. W. Otto, E. W. Felten, “Large-Step Markov Chains for the Traveling Salesman Problem,” Complex Systems, No. 5, pp. 299-326, 1991
[31] M. Padberg, G. Rinaldi, “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems,” SIAM Review, Vol. 33, No. 1, pp. 60-100, 1991.
[32] K. Mak and A. Morton, “A modified Lin-Kernighan Traveling Salesman Heuristic,” ORSA Journal on Computing, Vol. 13, p. 127, 1992.
[33] D.S. Johnson and L.A. MacGeoch, “The Traveling Salesman Problem: A Case Study In Local Optimization,” E.H. Aarts and J.K. Lenstra (eds), New York: Wiley and Sons, 1996.
[34] C. Rego, “Relaxed Tours and Path Ejections for the Traveling Salesman Problem,” To appear in the European Journal of Operational Research, 1996.

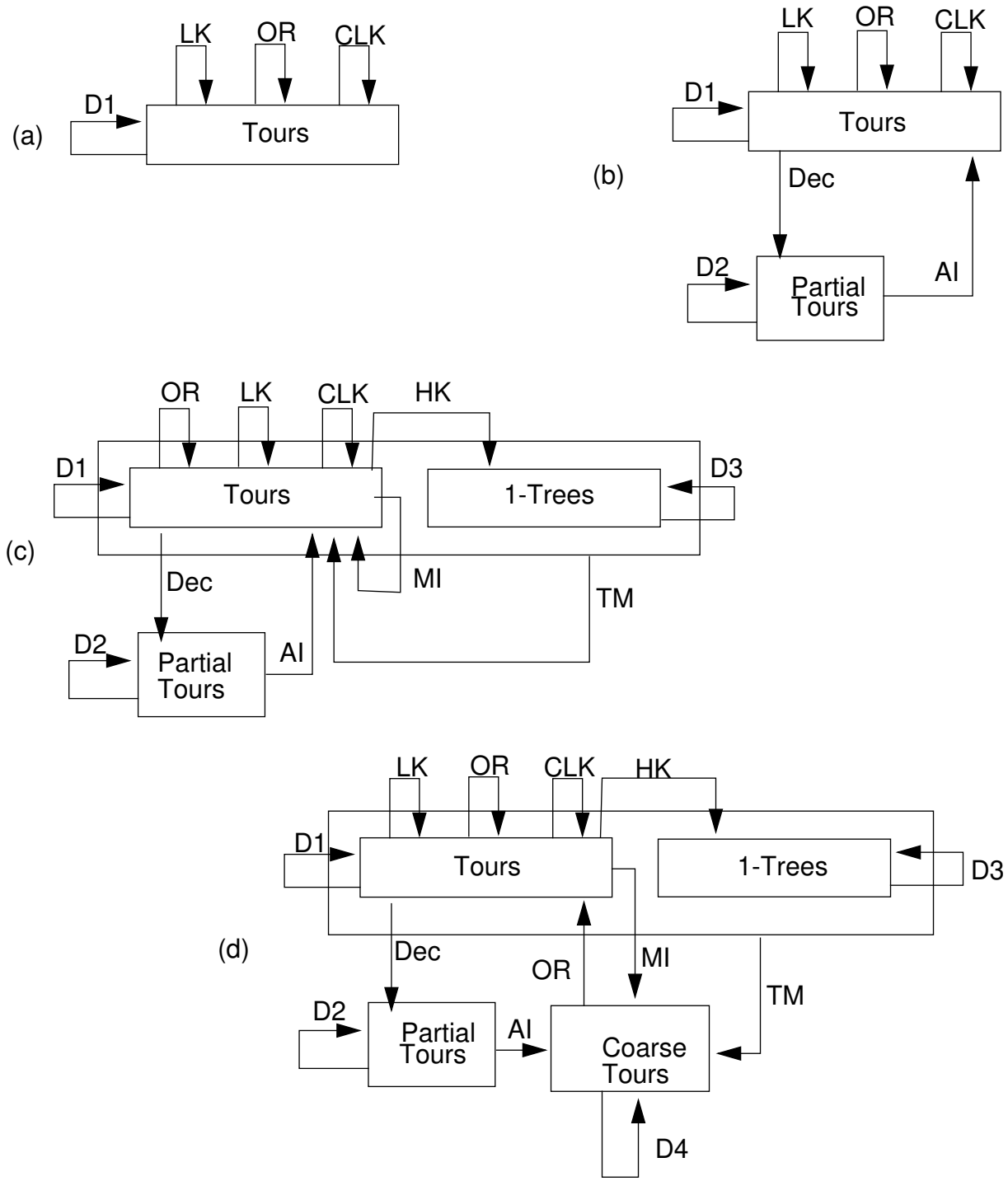

Fig. 1: Four data-flows for the traveling salesman problem. D1-D4 are destruction agents. The other agents (LK, OR, CLK, etc.) are construction agents built from the algorithms listed in Fig. 2.

LK	Lin-Kernighan, a long and powerful heuristic for the TSP [12]
CLK	a shorter and less powerful version of LK [14]
OR	Or-Opt, a moderately complicated, moderately powerful heuristic [14]
AI	Arbitrary Insertion, a very short and simple heuristic [14]
HK	the Held-Karp algorithm modified to convert tours into 1-Trees [13], [14]
Dec	a deconstructor that produces a partial tour from the common edges of two complete tours [14]
MI	a mixing heuristic that combines two tours to get one [14]
TM	a mixing heuristic that combines a tour with a 1-tree to give a new tour [14].

Fig. 2: A sample of algorithms for the traveling salesman problem.

DATA FLOW (See Fig. 1 for details)	PERFORMANCE Δ : the average difference in length between the best tour that was found in each of 15 runs and the optimum tour. $T_{\delta m}$: the average computation time per run with all the agents sharing one computer (a DEC 5000)							
	Krolak 24 100 cities [21]		LK 318 318 cities [14]		PCB 442 442 cities [22]		ATT 532 532 cities [23]	
	Δ (%)	$T_{\delta m}$ (sec)	Δ (%)	$T_{\delta m}$ (hrs)	Δ (%)	$T_{\delta m}$ (hrs)	Δ (%)	$T_{\delta m}$ (hrs)
(a)	0	35	1.27	2.9	1.20	4.2	0.87	7.5
(b)	0	39	1.13	2.4	0.89	3	0.47	6.8
(c)	0	39	0.06	1	0.26	4.8	0.40	14
(d)	0	13	0	1.5	0.01	3.5	0.06	13

Fig. 3: Results from applying the data flows of Figure 1 to four TSP problems. The results are averages over 15 runs. Each run was terminated when improvements in the tours ceased. All the agents of each data flow were made to share a single computer (a DEC 5000).

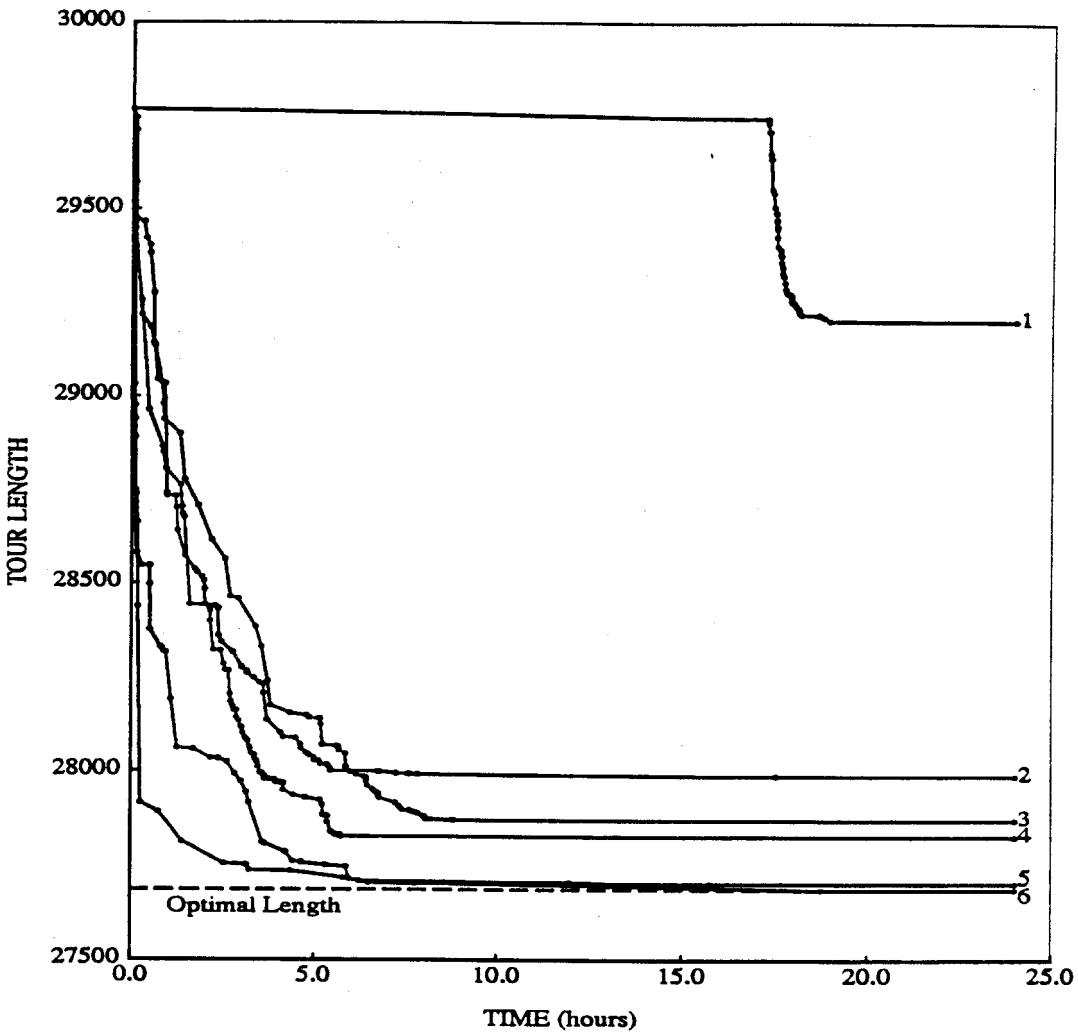

Fig. 4: Results from the application of six A-Teams to the ATT 532 traveling salesman problem. Each team was run on a single computer. The teams use the same node set (that of Fig. 1 (d)) but different sets of construction agents, as indicated below:

1: MI
2: MI, Dec, AI
3: MI, Dec, AI, HK, TM
4. MI, Dec, AI, HK, TM, CLK
5: MI, Dec, AI, HK, TM, CLK, OR
6: MI, Dec, AI, HK, TM, CLK, OR, LK

Notice that he larger teams are faster and produce better solutions than the smaller ones. Though not shown in the figure, the teams also produce better solutions than the agents from which they are composed [14].

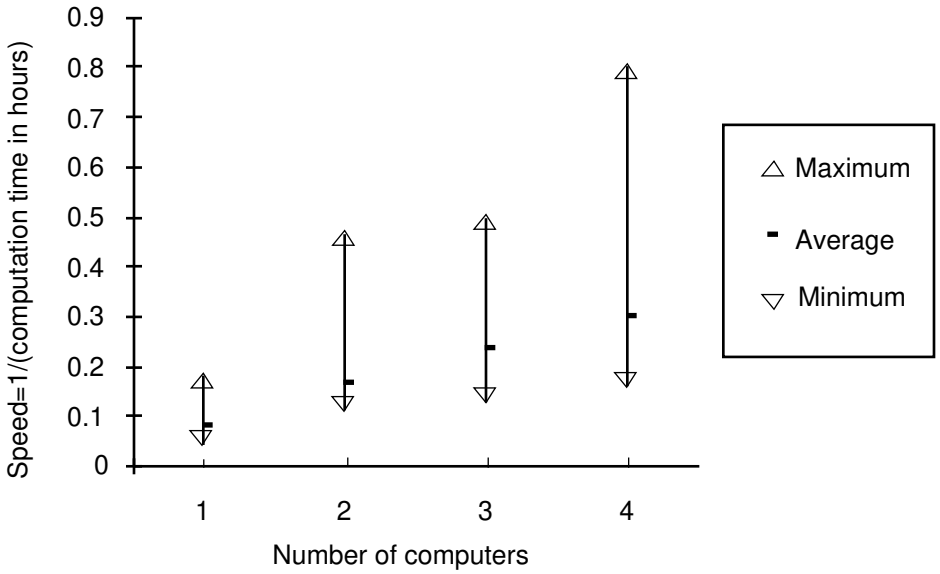

Fig. 5: Plots of speed vs. number of computers for problem ATT532 and the data flow of Fig. 1 (d). The average, maximum and minimum speeds were for 15 runs.

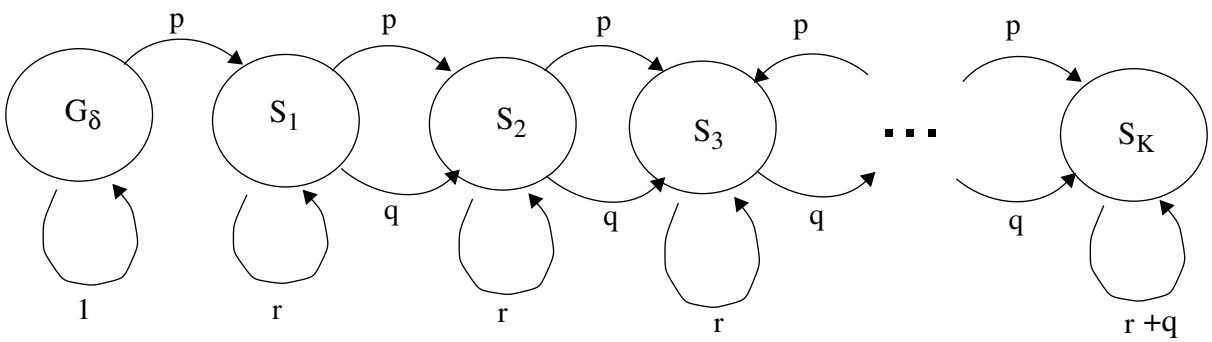

Fig. 6: A Markov chain. Nodes represent solutions states, arcs represent transition probabilities. Consider a trial solution in state S_n. The next agent to work on this solution has a probability p of converting the solution to a solution in S_{n-1}, a probability q of converting it to state S_{n+1} and a probability r of leaving its state unchanged

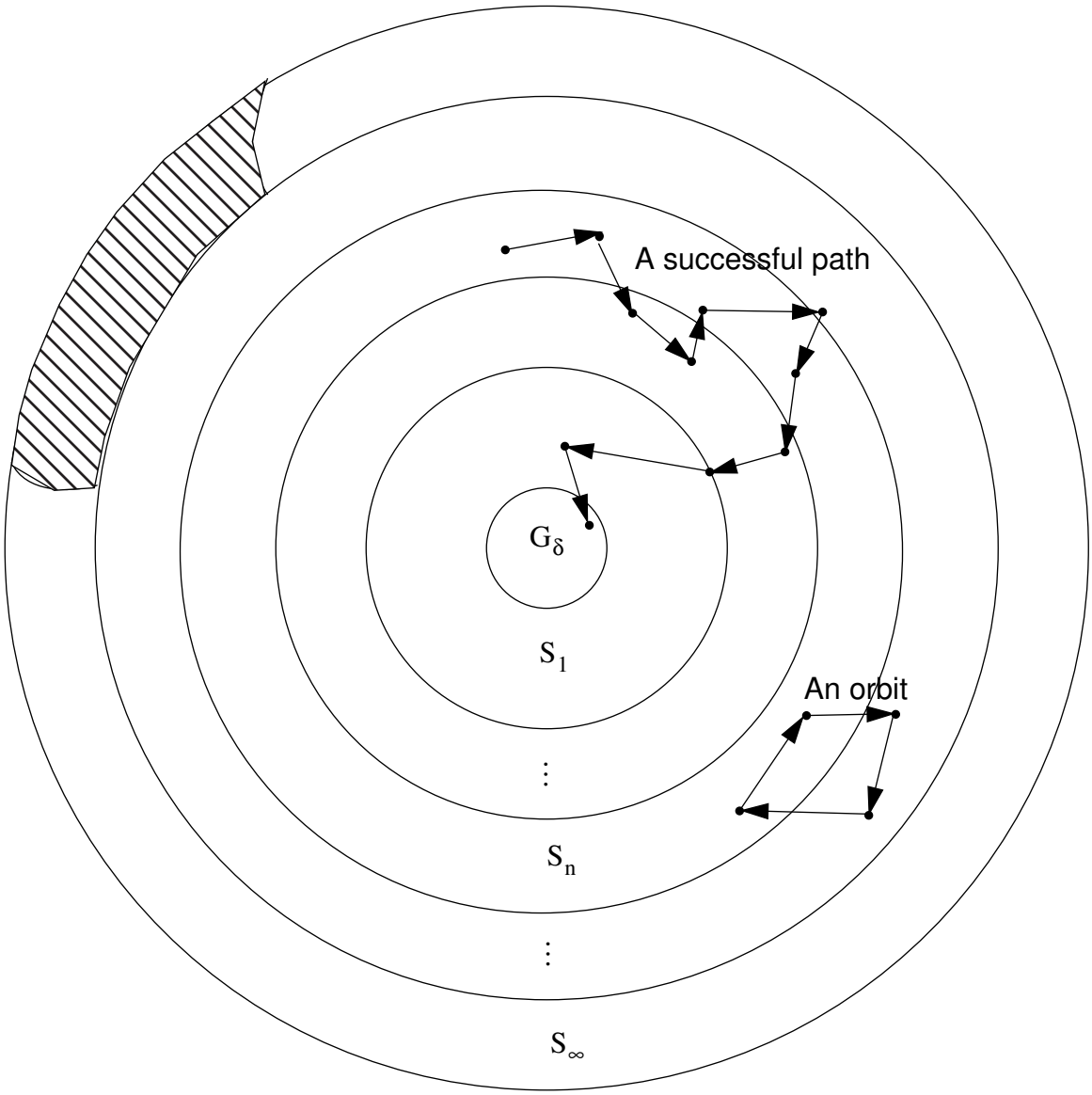

Fig 7: P(S, C, D, δ), the space of solutions partitioned into regions so that all the solutions in S_n are n-constructive operations from the goal space, G_δ, and all the solutions in the hatched region have been made inaccessible by destroyers.

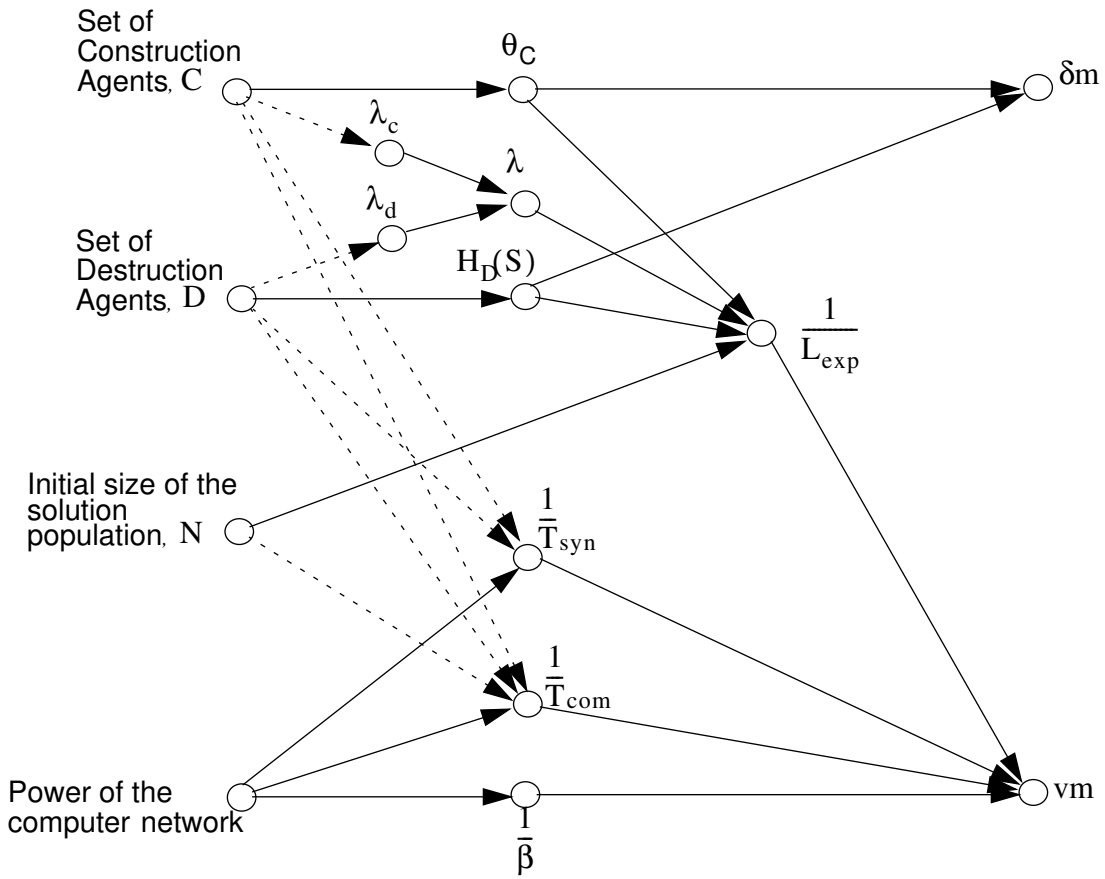

Fig. 8: Causal relations of a CDM. Each solid arc denotes a monotonically-increasing relationship. For instance the solid arc between N and $\frac{1}{L_{exp}}$ means that $\frac{1}{L_{exp}}$ increases monotonically with N.

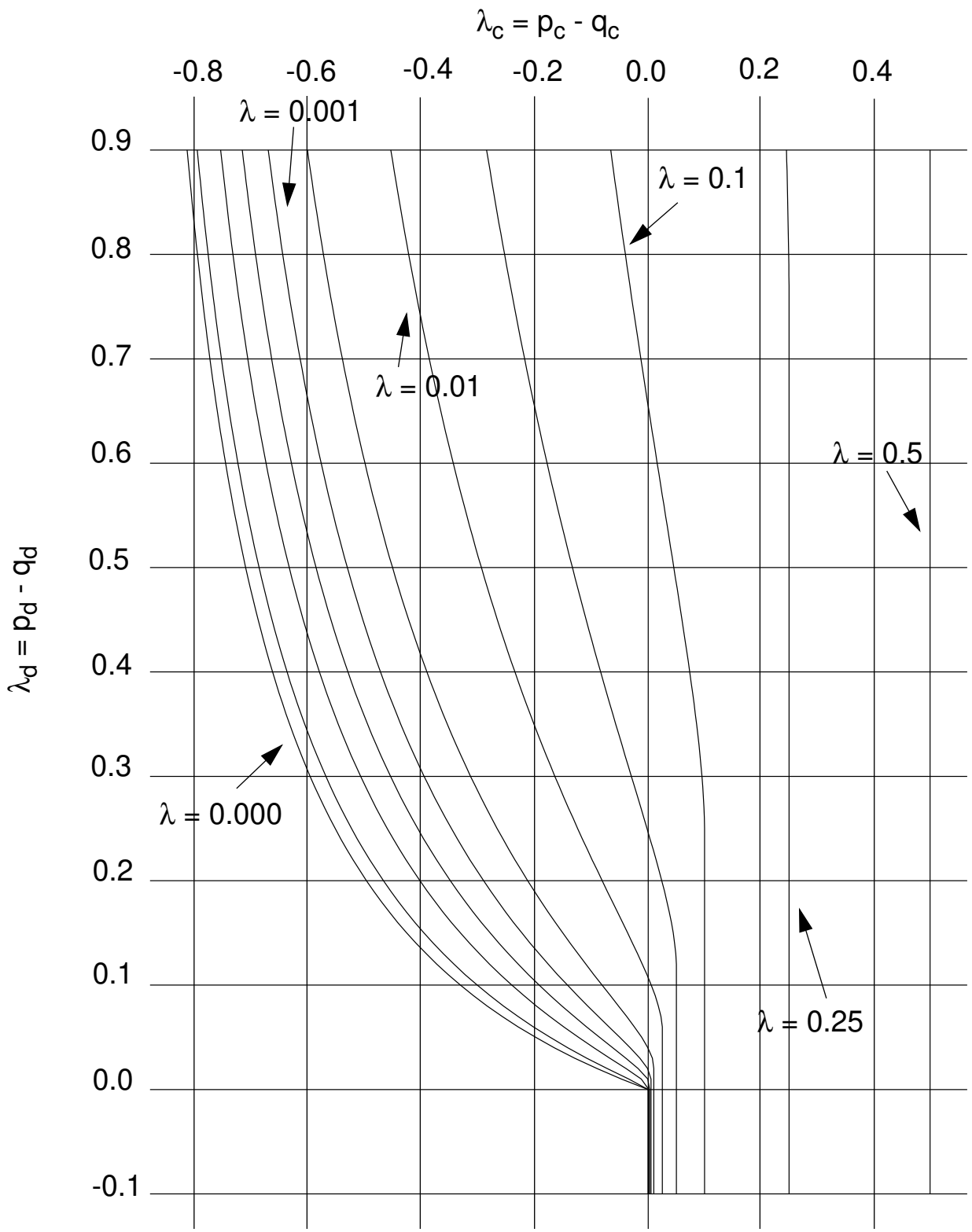

Fig. 9: Iso-drift curves for one set of values of q_d, r_d and r_c. Notice how λ, the overall drift depends on λ_c, the construction drift, and λ_d, the destruction drift.

APPENDIX

In this appendix we explore CDMs in greater detail.

Definitions:

Let:

C be the set of construction agents that acts on a CDM

θ_C be the set of algorithms contained in C.

D be the set of destruction agents that acts on a CDM

S be the space (set) of all possible solutions, good and bad, that can be stored in a CDM.

δ be an indicator of solution-quality such that δ increases as solution-quality increases.

G_δ be the subset of S that contains all the solutions of quality δ and better.

N be the size of the initial population of solutions stored in a CDM. (In real problems, N is always small in comparison to the size of S.)

T_δ be the expected amount of time for the population of solutions to evolve at least one solution of quality δ or better. G_δ is said to be reachable if T_δ is finite.

δm be the greatest value of δ such that G_δ is reachable.

$vm = 1/T_{\delta m}$ be the expected speed with which $G_{\delta m}$ is reached.

d(y) be the distance of y from G_δ, where y is any solution in S, and d(y) is the minimum number of construction-operations needed to convert y into a member of G_δ.

S_n be the subset of S containing all the solutions that are at a distance of n from G_δ, as in Fig. 6.

H be the power set of S (the family of all the subsets of S).

H_D be the subset of H that are recognized and erased y the destroyers in D.

p, q and r be the constant probabilities that the latest edge in any developing path will be a progressive, regressive or neutral edge, respectively; where a progressive edge moves the path's end closer to G_δ, a regressive edge moves it further away and a neutral edge leaves it at the same distance.

p_c, q_c and r_c be the values of p,q and r when the destroyers are disabled.

p_d, q_d and r_d be the conditional probabilities that a regressive edge, if considered for destruction, will be destroyed; that a progressive edge, if considered for destruction, will be destroyed; and that a neutral edge, if considered for destruction, will be destroyed.

$\lambda = p-q$ be the overall drift of the CDM; $\lambda_c = p_c - q_c$, be the drift of the constructors; and $\lambda_d = p_d - q_d$ be the drift of the destroyers.

β be the amount of time required for each agent to take one action.

T_{syn} and T_{com} be the expected synchronization and communication delays experienced by agents in developing a complete path (one that reaches $G_{\delta m}$).

Calculating the overall drift λ of the system.

Consider a single path through S as it is developed by the constructors and destroyers.

We assume that in each step a constructor which will extend the path is chosen with probability x and a destroyer which may shorten the path is chosen with probability 1-x. λ can then be computed from x, p_c, q_c, r_c, p_d, q_d and r_d.
Let:

e denote the edge most recently added to the path.

k_{pro}, k_{reg}, k_{neu} denote the probabilities that e is destroyed some time in the future given that it is a progressive, regressive or neutral edge respectively.

The most recently added edge e can be destroyed in two ways. Either it is destroyed before any other edge is added to the path, or it is destroyed after some other edge has been added. In the latter case the new edge has to be destroyed before e can be considered for destruction again. If we assume the A-team will run for a very large number of iterations, then, once the new edge is destroyed, the system will be in exactly the same state as when e was first added. Hence we get the following recursive formulae for k_{pro}, k_{reg} and k_{neu}.

$$k_{pro} = (1-x)q_d \sum_{n=0}^{\infty} (1-x)^n (1-q_d)^n + \left(1 - (1-x)q_d \sum_{n=0}^{\infty} (1-x)^n (1-q_d)^n\right)(p_c k_{pro} + q_c k_{reg} + r_c k_{neu})k_{pro}$$

$$= \frac{(1-x)q_d}{1-(1-x)(1-q_d)} + \left(1 - \frac{(1-x)q_d}{1-(1-x)(1-q_d)}\right)(p_c k_{pro} + q_c k_{reg} + r_c k_{neu})k_{pro}$$

$$k_{reg} = \frac{(1-x)p_d}{1-(1-x)(1-p_d)} + \left(1 - \frac{(1-x)p_d}{1-(1-x)(1-p_d)}\right)(p_c k_{pro} + q_c k_{reg} + r_c k_{neu})k_{reg}$$

$$k_{neu} = \frac{(1-x)r_d}{1-(1-x)(1-r_d)} + \left(1 - \frac{(1-x)r_d}{1-(1-x)(1-r_d)}\right)(p_c k_{pro} + q_c k_{reg} + r_c k_{neu})k_{neu}$$

The overall drift of the A-team then becomes

$$\lambda = x(p_c(1-k_{pro}) - q_c(1-k_{reg}))$$

Fig. 9 shows λ as a function of p_c, q_c, r_c, p_d, q_d and r_d for optimal x.

Reachability of G_δ

If G_δ is small or the set of construction agents is weak, not all solutions in S are at a finite distance from G_δ. Let S_∞ be this set of points for which there is no path to G_δ. Clearly we cannot guarantee reaching G_δ unless the destroyers make S_∞ inaccessible.

If:

- λ is positive, and
- if the outermost regions of S are either empty or made inaccessible by the destroyers,

that is, if there is a finite K such that for $k > K$, $S_k = \varnothing$ or $S_k \subseteq H_D$

then:

- G_δ is reachable.

proof:

Let j be the expected number of steps required to get one step closer to the goal.
Then

$$j = 1 + p(0) + r(j) + q(2j) \Rightarrow j = \frac{1}{p-q} = \frac{1}{\lambda} \quad (0)$$

so when $\lambda > 0$ we get a finite j. Starting with a solution in S_n the expected number of steps required to get to G_δ is $\frac{n}{\lambda}$. let E[n] denote the expected distance to G_δ from the randomly seeded solution. Since all solutions not made inaccessible by the destroyers are at distance at most K we have $E[n] < \infty$. Hence then expected time to goal from the randomly seeded solution is $\frac{E[n]}{\lambda} < \infty$.

Monotonic relationships. (Fig. 8)

1. Solution Speed v_m.

Let:

U be the subset of S that is not in H_D.

R_n be the residue of S_n, that is, the fraction of points in U that are at distances of n or greater from G_δ. In other words:

If:

- the destruction agents make the portion of S that is outside U completely inaccessible, preventing paths in U from ever leaving it;

• N starting points are randomly chosen from U, all points in U being equally likely;

• the best (closest to G_δ) of these points is identified and a path from it to G_δ is developed by the sequential application of construction agents and destroyers;

Then:

$$R_n = 0 \text{ if and only if } S_n \cap U = S_{n+1} \cap U = \ldots = \varnothing \quad (1)$$

$R_1, R_2, R_3,...$ decrease monotonically as the variety of constructive skills in increases, that is, as the number of agents in C increases (2)

$$n_{min} = \sum_{n=1}^{\infty} R_n^N \quad (3)$$

$$L_{exp} = \frac{1}{\lambda} n_{min} \quad (4)$$

where n_{min} is the expected distance of the best starting point from G_δ, and L_{exp} is the expected length of the path from this point to G_δ.

Proof:

Result (1) is obvious from the definition of R_n. Result (2) follows directly from the definition of R_n and the fact that S_n decreases as the skills in θ_C increases.

To see (3) note that the probability that the best of N starting points is in distance n from the goal equals the probability that all points are at least in distance n and not all points are at least in distance n+1 so

$$n_{min} = \sum_{n=0}^{\infty} n(R_n^N - R_{n+1}^N) = \sum_{n=1}^{\infty} nR_n^N - \sum_{n=1}^{\infty} (n-1)R_n^N = \sum_{n=1}^{\infty} R_n^N$$

(4) follows immediately from (3) and (0)

Now when N increases each term in the sum (3) decreases so the expected speed $v_m = \frac{L_{exp}}{n_{min}}$ with which we reach G_δ increases.

Likewise by (2),(4) an increase in θ_C or λ also causes the v_m to increase.

2. Solution-quality, δ_m.

S_∞ depends on δ and θ_C. Specifically S_∞ fills as δ increases, and empties as θ_C expands. Suppose that an improvement of solution quality from δ to δ-$\Delta\delta$ causes S_∞ to fill by the amount ΔS_∞. Then sufficient conditions for achieving this improvement are: an expansion of θ_C to empty part of ΔS_∞, and an expansion of H_D to contain the rest of ΔS_∞, all while maintaining $\lambda > 0$.

Age Based Restarting.

For small λ, the expected path length to the goal L_{exp} becomes unmanageably large. Faster convergence to the goal can then often be achieved by terminating all paths and restarting with a new random population of solutions. The decision to terminate a path should be made on the basis of information actually available. One piece of information that can readily be used is the number of operations performed since last restart.

Let $\Pi^0 = (\Pi_0^0, \Pi_1^0, \ldots)$ be the distribution for the best member in the set of initial solutions, that is $\Pi_n^0 = \text{prob}[\text{ best initial solution} \in S_n]$, and let

$$P = \begin{bmatrix} 1 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ p & r & q & 0 & \ldots & \ldots & \ldots & \ldots \\ 0 & p & r & q & 0 & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix}$$

Then $\Pi^N \equiv \Pi^0 P^N$ is the distribution of distance to goal of the solution after it has been worked on for N iterations. If after N iterations we haven't reached the goal state the conditional distribution of the distance to the goal is given by

$$\Omega_n^N = \Pr\left[\text{sol in } S_n(\theta, \delta) \text{ after N iterations} \middle| \text{ sol not in } G_\delta \text{ after N iterations}\right] = \frac{\Pi_n^N}{(1 - \Pi_0^N)}$$

and we may say that the solution has age N.

The average remaining number of iterations to get a solution of age N into G_δ is then

$\sum_{n=1}^{\infty} \frac{n}{p-q} \Omega_n^N$ For certain values of p, q, r and Π^0, even for some $p > q$ this remaining number of iterations will for some N exceed the expected time to reach the goal from the initial solution. For these values of p, q, r and Π^0 it is beneficial to use age based restarting. Suppose we always reseed the memory after N iterations. Then the expected number of iterations to reach G_δ is

$$N(\text{ exp. \# of restarts }) + \left(\text{ exp. \# of iterations to reach } G_\delta \middle| \ G_\delta \text{ reached in} \le N \text{ iterations }\right)$$

(*)

As a numerical example take , , and assume that the best newly created solution will always start in S_{10}