Asynchronous Teams: Cooperation Schemes for Autonomous, Computer-Based Agents

Sarosh Talukdar, Lars Baerentzen,
Andrew Gove and Pedro de Souza

EDRC18-59-96

ASYNCHRONOUS TEAMS: COOPERATION SCHEMES FOR AUTONOMOUS AGENTS

Sarosh Talukdar
Lars Baerentzen
Andrew Gove
Pedro de Souza

Carnegie Mellon University
Pittsburgh, PA 15213

ASYNCHRONOUS TEAMS: COOPERATION SCHEMES FOR AUTONOMOUS, COMPUTER-BASED AGENTS

Sarosh Talukdar Lars Baerentzen 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 network of autonomous agents (workers) and memories (workplaces). Results (trial solutions to computational problems) circulate continually through this network. Agents work in parallel and cooperate by modifying one another's results. We have accumulated a good deal of experience in making the circulating results converge to better solutions of optimization and constraint satisfaction problems than the agents can find when working independently.

This paper does three things. First, it distills our experiences with A-Teams into a protocol for designing them. Second, it points out that a sufficient condition for the circulating results to converge is that the skills of the agents that construct new results be complementary to the skills of the agents that destroy old results. Third, it argues that this complemantarity is relatively easy to achieve. The practical implications are: a) the quality of solutions obtained by any problem-solving-algorithm, even the best one available, can invariably be improved by combining it with other available algorithms into an A-Team, b) the quality of solutions obtained by any A-Team can invariably be improved by expanding its size, and c) expansions are relatively easy to make.

I. INTRODUCTION

None of the known algorithms for optimization and constraint satisfaction is without weaknesses-the rigorous algorithms tend to be too slow, the heuristics, too unreliable. This situation is likely to continue into the foreseeable future. To make the best of it, we have been seeking ways by which algorithms can be made to cooperate, so together they can 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].

> Definition: an A-Team is an evolving sequence of strongly cyclic data flows. A data flow is a directed hypergraph like those shown in Fig. 1. Nodes in a data flow are Venn diagrams representing complexes of overlapping memories, or more precisely, the objects these memories can contain. Arcs in a data flow represent autonomous agents that read from the memories at their tails and

write to the memories at their heads. A data flow is strongly cyclic if all, or almost all, its arcs are in closed loops.

Each A-Team is dedicated to one problem. Each data flow is a distinct scheme for solving instances of this problem and is implemented in a network of computers. Results (trial-solutions) accumulate in the memories of the data flow (just as they do in blackboards) to form populations (like those in genetic algorithms). These populations are continually modified by two types of agents: construction agents that add members to the populations and destruction agents that eliminate members from the populations. The latter work from lists of solutions 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 every data flow of 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 centralized control system to get in their way).

Agents cooperate by working on one another's results. Because the agents are autonomous, 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.)

Since every agent decides for itself what, if anything, to do and when, if ever, to do it, and since every agent knows nothing more about its collaborators than the results they produce, one might think that the agents would tend to work at cross purposes and the entire organization be prone to anarchy. Surprisingly, this not always the case. 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 mill scheduling [17], paper mill scheduling [25], [26], train-scheduling [15], and con-

straint satisfaction [18]. Not only do the data flows of these A-Teams produce very good solutions, but they appear to be scale-effective, that is, the data flows can always be made to produce even better solutions by the addition of certain agents and memories. (Scale-effective organizations are embodiments of the proverb "two heads are better than one." Though this proverb is old, scale-effectiveness is rare in synthetic organizations which are better described by another proverb, "too many cooks spoil the broth." In a scale-effective organization there can never be too many "cooks/' at least not from the perspective of product-quality.)

Knowing that an organization is scale -effective is of great practical significance. Specifically, the problem of improving solution-quality in 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.

II. AN ITERATIVE DESIGN PROTOCOL

For several years we have been experimenting with a 6-step protocol for designing A-Teams. Each step involves choices for which we have not been able to devise automatic procedures. For making these choices we offer the reader some very general, experience-based advice, illustrated with the choices we made in building an A-Team for one version of the traveling salesman problem (TSP).

The TSP is a a prototypical combinatorial optimization problem. Its simplest form is: given M cities and their separations, find the shortest tour of the cities. (A tour is a closed path that goes through every city.) The number of distinct tours grows so rapidly with M that it is impossible to conduct an exhaustive search for the shortest tour, even when there are as few as 30 cities. Practical instances of the TSP often have hundreds and sometimes thousands of cities. Rigorous algorithms that can solve large TSPs in reasonable (polynomial) amounts of time are unknown. However, there are dozens of faster heuristics, many of which are readily availabe as computer codes. The best pre-coded heuristic we were able to obtain was the Lin-Kernighan (LK) algorithm [12]. This algorithm tries to improve any tour it is given, and often succeeds. However, once improved, a tour cannot be further improved by LK. For large problems, like the 532 city problem of [23], the "improved tour" tends not to be optimal unless the "given tour" is close to optimal. The A-Team we built improves on the performance of LK by combining it with several lesser algorithms.

Step 1. Form a superproblem.

The term "superproblem," as used here, means any set of problems that includes the problem-to-be-solved. The members of the superproblem need not be disjoint, nor even distinct. Some may be components of the problem-to-be-solved, others may be larger and more complex than it. But they must be such that they can be connected through agents into closed loops so that trial-solutions of each problem can be constructed from trial-solutions of the preceeding problem in the loop.

The superproblem we chose for the TSP contains four members: find an optimal tour,

find good tours, find good partial tours, and find good 1-trees; where "good" means "containing many of the arcs of an optimal tour" and a 1-tree connects all the cities but does not always form a tour (the 1-trees form a superset of the tours).

Step 2. Assign 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 the marginal benefits fall off rapidly. Therefore, moderately sized populations can be expected to work as well as very large ones.

Even moderate populations can occupy a great deal of storage space, especially when each trial -solution is a complex object, such as a building-design. In such cases, the representation should be a carefully chosen compromise between compactness and clarity.

For the TSP, we choose M, the number of cities, as the population size (M/2 and 2M seem to work equally well) and 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. Select a set of algorithms or operators for each member of the superproblem.
The greater the range of skills of these algorithms, the better the solutions 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 a state-of-the-art-branch-and-bound-algorithm, when used alone, will invariably do less well than a mix of considerable variety and range.

We choose the set of algorithms shown in Fig. 2 for the TSP.

Step 4. Form each algorithm into an autonomous agent.
Think of an agent as operating between two worlds-one it percieves, the other it affects. These worlds may overlap. If the agent is computer-based, these worlds can always be thought of as input and output memories for the agent [19]; and the agent can always be decomposed into three components [19]: a communication system, an operator and a control system. The communication system connects the agent to its input and output memories. The operator modifies objects obtained from the input memory. The control decides which objects will be obtained and when, that is, the control system consists of selectors (to choose objects from the input memory for the operator to work on) and schedulers to determine when the operator will work and which of the available computers it will use.

> 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.

The key to effective cooperation among autonomous agents is in the design of their selection strategies. Where the quality of the solution can be measured by a single attribute (such as tour length for the TSP), a very simple selection strategy seems to work quite well. This strategy is a mirror image of the solution- rejection-strategy used in simulated annealing, specifically: select solutions randomly with a bias towards the better solutions. Murthy [9] describes a variant of this strategy 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; arrange for the probability of selection to increase as the magnitude of the angle between these 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. *Form autonomous destroyers.*
Agents can be of two types: constructors (that add trial-solutions to their output-memories) and destroyers (that erase trial-solutions from their output-memories). The agents produced in the previous step are constructors. Their actions must be balanced by destroyers or the memories would soon become clogged with trial-solutions.

Destroyers can serve two additional functions. First, by quickly erasing any solutions that fall in certain parts of their output spaces (the sets of all the solutions that can be stored in their output-memories), destroyers can make those parts almost inaccessible to the constructors. (For a region to be truly inaccessible, results would have to be prevented from falling in it, not merely erased after falling in it.) Second, destroyers can terminate undesirable patterns of constructor behavior, such as repeating sequences of solutions, by recognizing and erasing them.

For the TSP, we used destroyers that would select tours for erasure randomly but with a strong bias for the longest tours (just as in the solution-rejection-strategy of simulated annealing). The destroyer were scheduled to act so there were always a few slots left openfor the constructors to fill.

Step 6. Form the agents and memories into a strongly cyclic data flow.
Implement this data flow in a network of computers. Seed the memories with initial populations of trial-solutions, activate the agents and monitor the changes in the solution-populations. If these changes are overly slow in converging, that is, if at least one complete solution of acceptable quality is overly slow in appearing, then repeat from step 3. If convergence is still too slow, repeat from step 1. In choosing the changes to make, we have found it best to proceed under the assumption that strongly cyclic data flows can easily be made scale-effective. This suggests beginning with a small data flow and then adding agents and memories until a data flow with adequate performance is obtained.

Four of the data flows we designed for the TSP are shown in Fig. 1. All these data flows find optimal solutions much more often than LK, the best algorithm they contain. For Instance, LK is seldom able to find the optimal solution of the 532 city problem of [23] while the data flows of Fig 1 invariably do.

Observed Behavior

Results for the TSP are shown in Figs. 3 and 4. Notice the effects of scale on solution-quality, especially, how quality improves with the size of the data flow. Notice also that the speed with which the final solution was obtained increased with the size of the data flow (Fig.3), even though all the agents and memories were made to share a single computer. When more computers were made available (Fig. 4), the larger data flows invariably finished faster than the smaller ones. Specifically, when four or more computers were provided to each data-flow, the largest data flow produced better solutions faster than any other. In other words, solution-quality and speed appear to be commensurate: both can be improved by the addition of certain agents and memories, provided enough computers are made available to allow the agents to work in parallel. Similar behavior has been observed over a variety of other optimization problems for which A-Teams have been developed [7-11, 14-18, 24-27]. More specifically, A-Teams seem to display the following problem-independent behaviors:

1. Diversity: solution-quality increases with the range of skills of the construction agents.
2. Scale: there is little, if any, penalty for an excess of construction skills. Rather, scale-effectiveness seems to be commonplace; solution quality can invariably be improved by the addition of construction and destruction agents.
3. Expansion: adding autonomous agents to strongly cyclic data flows is relatively easy, regardless of whether the agents are large or small, general or specialized.
4. Duality: adept destruction can compensate for inept construction, and vice-versa.
5. Population size: solution-quality benefits from increasing the sizes of solution-populations, but these benefits are prone to saturation.
6. Parallelism: solution-speed improves as computers are added until there are enough computers for all the agents to work in parallel all the time. Often, the speed-up is near-linear.

III. COOPERATION IN STRONGLY CYCLIC DATA FLOWS

What mechanisms are responsible for the problem-independent behaviors of A-Teams? What are the underlying phenomena and causal relations? We will tackle these questions with the aid of a device, called a CDM (constant drift memory) which, it can be argued, is a conceptually useful model of any A-Team memory. The argument, in outline, is:

- all forms of cooperation can be modeled 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;
- most, if not all, cyclic memories can be modeled by a particularly simple type of cyclic memory called a CDM.

The technical apparatus needed to make this argument is outlined below and detailed in the Appendix.

Definitions

Define cooperation as any exchange of data among agents, regardless of whether the exchange is productive or not. If the agents are computer-based, then all the different ways in which they can exchange data can be represented by data-flows [19].

Consider the data flow of a problem-solving organization. Each memory in this data flow is dedicated to some subproblem of the overall problem to be solved, and contains a population of trial-solutions to this subproblem. The dynamics of these populations are determined by their initial values and by the agents that act on them.

Define:

- the effectiveness of a memory to be the double: ($5m_f$ vm), where 5m is the highest quality solution that will appear in the memory, and vm is the expected speed of this appearance.
- a memory to be a primary memory if its subproblem is the same as the overall problem.
- a memory to be a cyclic memory if all the agents that write to it also read from it. (Note: any memory in a strongly cyclic data flow is well approximated by a cyclic memory. After all, no memory can see any more of its containing data flow than the 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.)
- a memory to be scale-effective if it is cyclic and there are agents that cause 8m to improve monotonically when these agents are added to the memory;
- a data flow to be scale-effective if at least one of its primary memories is so.
- a cyclic memory to be a constant drift memory (CDM) if the dynamics of each of its trial-solutions are described by a Markov chain of the sort shown in Fig. 5.

Some of the more important structural features of a CDM are:

- The members of the initial population of solutions in a CDM are chosen randomly from the space of all possible solutions that can be stored in the memory. 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 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 CDM and its agents are implemented in a distributed 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.

Consider any CDM. Let:

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

6_C be the set of operators contained in C.

D be the set of destruction agents that acts on the memory

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

5 be an indicator of solution-quality, such that S increases as solution-quality increases.

Gs be the subset of S that contains all the solutions of quality 5 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§ be the expected amount of time for the population of solutions to evolve at least one solution of quality 5 or better. G§ is said to be reachable if T_6 is finite.

5m be the greatest value of 8 such that G§ is reachable,

vm = 1/Tgm be the expected speed with which G^ is reached.

d(y) be the distance of y from Gg, where y is any solution in S, and d(y) is the minimum number of operations needed to convert y into a member of G5.

P(S) be a partition of S into regions S_o, S^..., Soo, such that S_n contains all the solutions that are at a distance of n from Gg, as in Fig. 6.

H be the set of all the paths in S.

F(H) be a fuzzy partition of H into regions of desirable and undesirable paths, as in Fig. 7

H_D be the subset of H that the destroyers can recognize and erase.

(3 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 Gg_m).

Consider Sj and s_{i+1}, the two latest points in any developing path. Let:

p, q and r be the probabilities that s_{i+1} is closer, further and at the same distance, respectively as Sj from G_5.

Pc» *Qc* and r c be the values of p,q and r when the destroyers are disabled.

Pd» Qd and r d ^ 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_s from G5.

λ(s_i) = p - q be the overall drift of the CDM at SJ; ^(SJ) = p_c - q_c, be the component of drift contributed by the constructors; and ^(Sj) = p_d - q_d be the component of drift contrib-

uted by the destroyers. (In a CDM, *X* is constant, but in other types of memories, *X* could vary over S.)

A(S) be the space of all the A.(Sj). Note: in a CDM$_f$ *X,* A^ and A$_{id}$ are constants for all points at finite and non-zero distances from the goal.

Convergence Conditions And Causal Relations

For any CDM, it can be shown (see the Appendix) that if:

- *X* is positive, and
- there exists a finite K such that $S_k = 0$ or S_k c H_D for all $k > K$,

then G§ is reachable.

In other words, any mix of agents that makes the outer regions of S either empty or inaccessible, and makes the drift at all accessible points positive, will produce a solution of quality 5 or better.

It can also be shown that the causal relations among the variables of a CDM are as depicted in Fig. 8.

CDM Behavior

The succeeding material applies the convergence conditions and causal relations of CDMs to determine and explain their behaviors along the six dimensions used earlier in describing the observed behaviors of A-Teams.

1. Diversity

P(S), the partition of the solution space (Fig. 6), can be reconfigured in two ways: by increasing 5 which causes a migration of points to the outer regions, and by increasing the number of construction operators which causes a migration in the opposite direction. The latter migration is strongest when the new construction operators contain new and powerful skills, in other words, the solution space contracts about a goal set as the range of skills of the construction operators increases. Notice that the addition of certain destroyers has a similar effect. Of course, destroyers cannot influence the individual distances of solutions from a goal set. But they can make the outer regions of the solution space essentially inaccessible by quickly erasing any solutions that happen to fall there. In effect, they can truncate the solution space, reducing the average distance of the remaining solutions from any goal set, and thereby, causing the accessible part of the solution space to contract about that goal set.

2. Scale

For a CDM to be scale-effective and for solutions of arbitrarily high quality to be reachable, there must be agents that: a) can be added to the CDM, b) make S^ empty or inaccessible, and c) leave the overall drift, *X,* positive for all points that are accessible and not in the goal.

The value of *X* depends on the values of X_c and *X^,* the drifts of the individual constructors and destroyers. These individual drifts are measures of selection acuity. A construc-

tor has a positive value for X_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 X_d when the its decisions to erase solutions are correct more often than wrong.

Expressions for the dependence of X on X_c and X_d can be found in the Appendix and visualized with the aid of curves of the sort shown in Fig. 8. This dependence is such that destruction has little effect on overall drift when the construction agents make relatively few selection errors ($X_c > 0.2$). But when construction agents are likely to make numerous selection errors, then destroyers can be used to erase the results of these errors, yielding a high overall drift.

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 there is the right mix of agents. One way to achieve such a mix is to: a) include as wide a range of construction knowledge (compiled into operators) as is available, in order to shrink the outer regions of the solution space, b) design and add destroyers to make whatever remains of the outer regions inaccessible, and c) add destroyers to erase the results of mistaken patterns of construction activity, and thereby make the overall drift positive.

3. Expansion

If agents are non-autonomous, some of their controls are packaged separately in a supervisory system. In such cases, the addition of a new type of agent usually requires a long and painful reengineering of this system. However, when each agent comes with its own complete control system, as is the case with autonomous agents, then no reengineering is 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.

Since the conditions for finding solutions of arbitrarily high quality in a CDM place no restrictions on agent type or granularity large agents can be mixed with small, and general agents with specialists.

4. Duality

In a CDM, construction and destruction are dual processes in the sense that strengths in one can compensate for weaknesses in the other. The evidence is as follows. First, the solution space can be made to contract about any goal set by adding constructors or destroyers. Second, the overall drift is sensitive to both (Fig 8). And third, the subgraphs connecting C and D to 5m and vm in the causal diagram (Fig. 7) are almost symmetric.

5. Population Size

The causal diagram (Fig. 7) indicates that increases in the size of the solution-population do not affect solution-quality but do benefit solution-speed. A more detailed analysis of these benefits (Appendix) shows that they are prone to saturation and affect only the early stages of path-development.

6. Parallelism

In a CDM, the addition of agents improves solution-quality, provided these additions leave the overall drift positive. The causal diagram of Fig. 7 indicates that such additions will also improve solution-speed making quality and speed commensurate instead of conflicting attributes, if the completely solid paths from C and D to quality and speed dominate the paths containing broken arcs.

Since synchronization delays are nonexistent for autonomous agents, the exclusive use of such agents causes the T_{syn} node to disappear from the causal diagram along with all the broken paths that go through it. In other words, the exclusive use of autonomous agents makes the conditions for commensuration much easier to meet. (In contrast, the performance of organizations with centralized control systems is often dominated by synchronization delays, making it necessary to pay for increases in solution-quality with decreases in solution-speed.)

CDMs As Models of More Complex Cyclic Memories

The observed behaviors of A-Teams are similar to the derived behaviors of CDMs. Are the mechanisms that produce these behaviors also the same? In other words, can CDMs be used to understand the internal workings of A-Team memories? We believe so.

Ofcourse, there are structural differences between A-Team memories and CDMs. Specifically, in an A-Team memory: a) an agent may use several old solutions to produce a new one, b) the overall drift, or rate of diffusion of solutions towards the goal, is unlikely to be constant, and c) the agents may have quite different computational times. We believe these differences are unimportant to solution-quality and can be made unimportant to solution-speed by scheduling strategies such as assigning larger computers to the larger agents. Empirical evidence in support of this belief is emerging in the form of devices, designed on the basis of insights obtained from CDMs, that work in real A-Teams. We feel that it is only a matter of time before theoretical evidence appears in the form of a dilation of the set of CDMs that preserves their behavior but eliminates the structural differences with A-Teams.

IV SUMMARY

A-Teams would appear to be useful for any problem with a rich set of imperfect algorithms, that is, any problem for which many algorithms are available but none is completely satisfactory. In function, an A-Team is a combination of algorithms that produces better solutions than any of the algorithms can when working independently. In structure, an A-Team is a sequence of increasingly complex data flows. Each data flow consists of autonomous agents and memories connected into a strongly cyclic network. Each agent contains one problem-solving algorithm. Each memory is dedicated to one problem. Collectively, the memories represent a superproblem or cover of the problem-to-be-solved, that is, a set of problems that includes the problem-to-be-solved.

The later data flows in an A-Team are obtained by modifying earlier ones, usually by the

addition of loops, that is, by expanding the superproblem and adding new agent.

In the operation of a data flow, trial-solutions to problems are produced by the agents and stored in the memories to form populations. Agents cooperate by working on one another's solutions.

Agents are of two types: construction agents that add solutions to populations and destruction agents that erase solutions from populations. In effect, the destroyers can make parts of a solution space inaccessible to constructors by quickly eliminating any solutions that happen to fall in them.

All the agents, whether constructive or destructive, are autonomous. Each such agent consists of an operator (algorithm), a selector and a scheduler. The skills of the agent to create or modify solutions are contained in its operators, the intelligence with which it applies these skills is contained in its selector and scheduler.

A-Teams are unusual among synthetic problem-solving systems, not because of any one feature, but rather, because of their combination of features. These include the use of superproblems rather than proper decompositions of the problem-to-be-solved, populations of solutions, autonomous constructors and destroyers, and strongly cyclic data flows. How can A-Teams be understood and made to work? One way is through the use of models called CDMs (constant drift memories).

A-Teams and CDMs display similar behaviors. There is reason to believe that their causal mechanisms are also similar. This being so, the conceptual aids and analytical results for CDMs can be applied to understand and better design the strongly cyclic data flows that constitute A-Teams.

The most useful of the conceptual aids are P(S), F(H), A(S) and the diagram of Fig.8. P(S) partitions all solutions by their distances from the goal; F(H) partitions all possible patterns of construction activity by their desirability; A(S) is a field of the net rates at which solutions will drift or diffuse towards the goal; and the diagram distinguishes monotonic causal relations from non-monotonic ones. As such, P(S) provides a view of what the set of construction-operators in a data flow can do, if they are controlled perfectly; F(H) provides a view of the mistakes the construction-operators can make, if they are controlled imperfectly; A(S) shows how well the constructors and destroyers will actually do when they are working together; and the causal diagram provides insights into how to improve performance.

The goal of the agents working on any memory in an A-Team, namely, finding a solution of acceptable quality in a reasonable amount of time, can be broken into two sub-tasks: a) make the outer regions of P(S), the solution space associated with the memory, empty or inaccessible, and b) make $V(s)$, the net drift at point s in the solution space, positive and as large as possible for all accessible values of s. Any mix of construction and destruction agents that covers these tasks will, if the agents are allowed to work on one another's results, reach the goal. That such mixes exist has been amply demon-

strated, at least for optimization problems. However, automatic procedures for synthesizing such mixes are still unavailable. Instead, there are only very general guidelines, of which the most important are:

- use autonomous agents whenever possible. The convergence conditions apply whether the agents are autonomous or not and it is sometimes convenient to mix the two types. But autonomous agents have no synchronization delays, and therefore, can provide dramatic speed-ups when allowed to work in parallel. In addition, they are easier to add to data flows than their non-autonomous counter parts.
- design agents to encapsulate the relevent knowledge in its naturally occurring chunks. In other words, put knowledge on what to do and where to search, into construction agents; knowledge on what to undo and where not to continue to search, into destruction agents; put large chunks of knowledge into large agents; small chunks into small agents.
- Include all the best and most powerful construction operators available. The resulting contractions in P(S) invariably exceed those produced by mixes that include only weak construction operators, such as crossover, or a single powerful operator, such as a state-of-the-art-branch-and-bound.
- use destroyers to increase the net drift by erasing poor solutions and ineffective patterns of construction. In other words, arrange for the destroyers to know, or to learn about, poor solutions and ineffective patterns.

V 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, 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, *Al 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] R 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 Department, 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] J. A. Lukin, A. P. Gove, S. N. Talukdar and C. Ho, "An Automated Probabilistic Method for Assigning Backbone Resonances of (13^C, 15^N)-Labelled

[18] S. R. Gorti, S Humair, R. D. Sriram, S. Talukdar, S. Murthy, "Solving Constraint Satisfaction Problems Using A-Teams,[M] 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, "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, 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

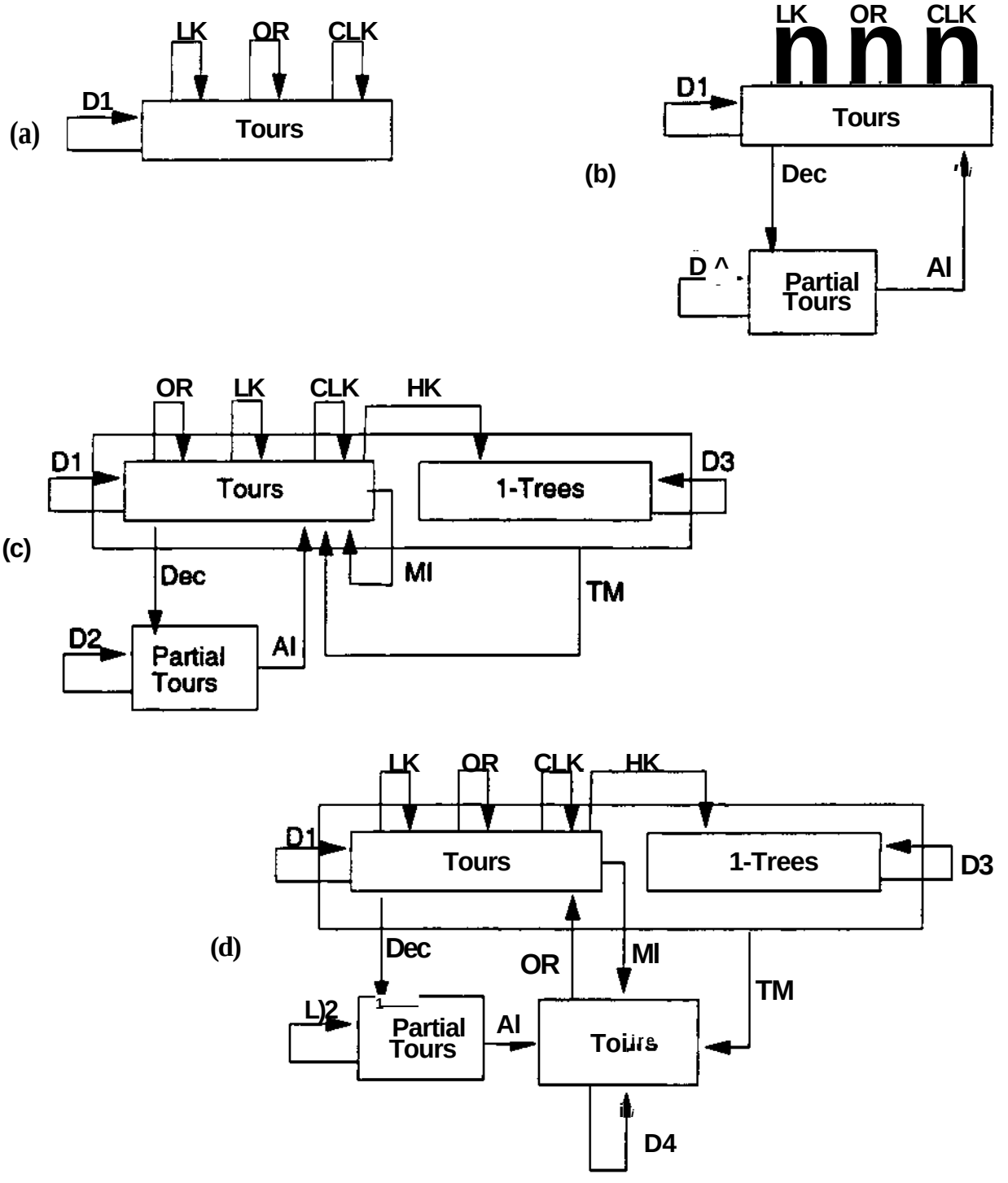

Fig. 1. Four data-flows from an A-Team for the traveling salesman problem. D1-D4 are destruction agents (they eliminate members from solution populations). The other agents are constructive (they add members to solution populations using the algorithms listed in Fig. 2).

LK	Lin-Kernighan, the longest, most powerful and by far the best TSP algorithm we could obtain [12]
CLK	a shorter and less powerful version of LK [14]
OR	Or-Opt, a moderately complicated, moderately powerful algorithm [14]
AI	Arbitrary Insertion, a very short and simple algorithm [14]
HK	Held-Karp, an algorithm for converting tours into 1-Trees [13]
Dec	a deconstructor that produces a partial tour from the common edges of two complete 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. 2. A sample of algorithms for the traveling salesman problem.

DATA FLOW (See Fig. 1 for details)	PERFORMANCE A: the difference in length between the best tour that could be found and the optimum tour. Ts_m: computation time with all the agents sharing one computer (a DEC 5000)							
	Krolak 24 100 cities [21]		LK318 318 cities [14]		**PCB** 442 442 cities [22]		**ATT** 532 532 cities [23]	
	A (%)	(sec)	A (%)	(hrs)	A (%)	**(hrs)**	A (%)	(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 2 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 A-Team were made to share a single computer- a DEC 5000.

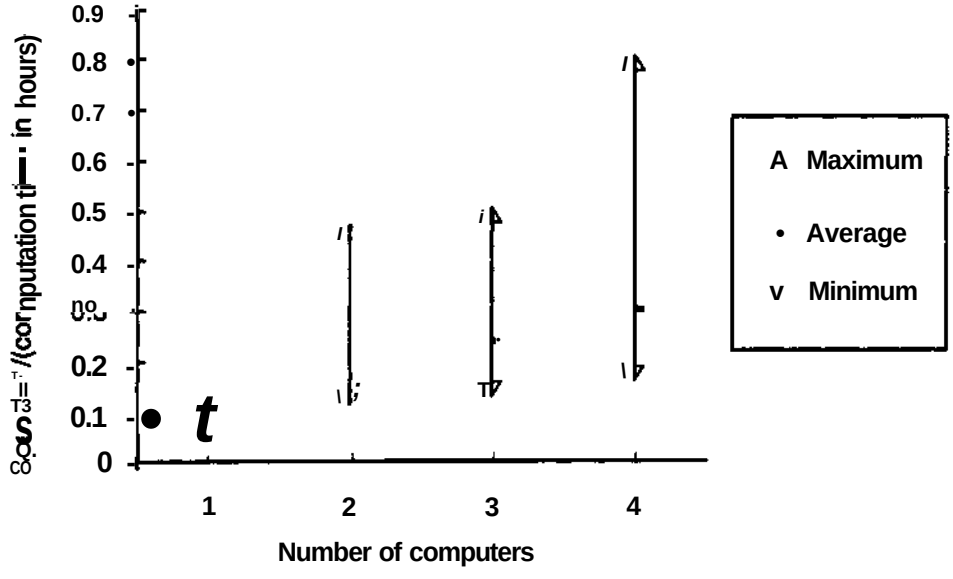

Fig. 4: Plots of speed vs. number of computers for problem ATT532 and the data flow of figure 1 (d). The average, maximum and minimum speeds were for 15 runs.

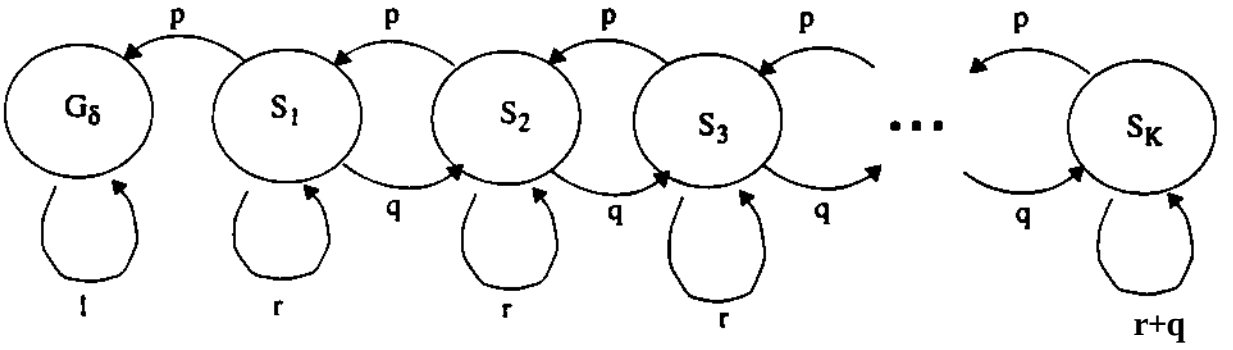

Fig. 5. A Markov chain. Nodes represent solutions states, arcs represent transition probabilities. Consider a trial solution in state S_n. The next agent to work on this solution has a probability p of converting the solution to a solution in S_{n_i}, a probability q of converting it to state S_{n+1} and a probability r of leaving its state unchanged

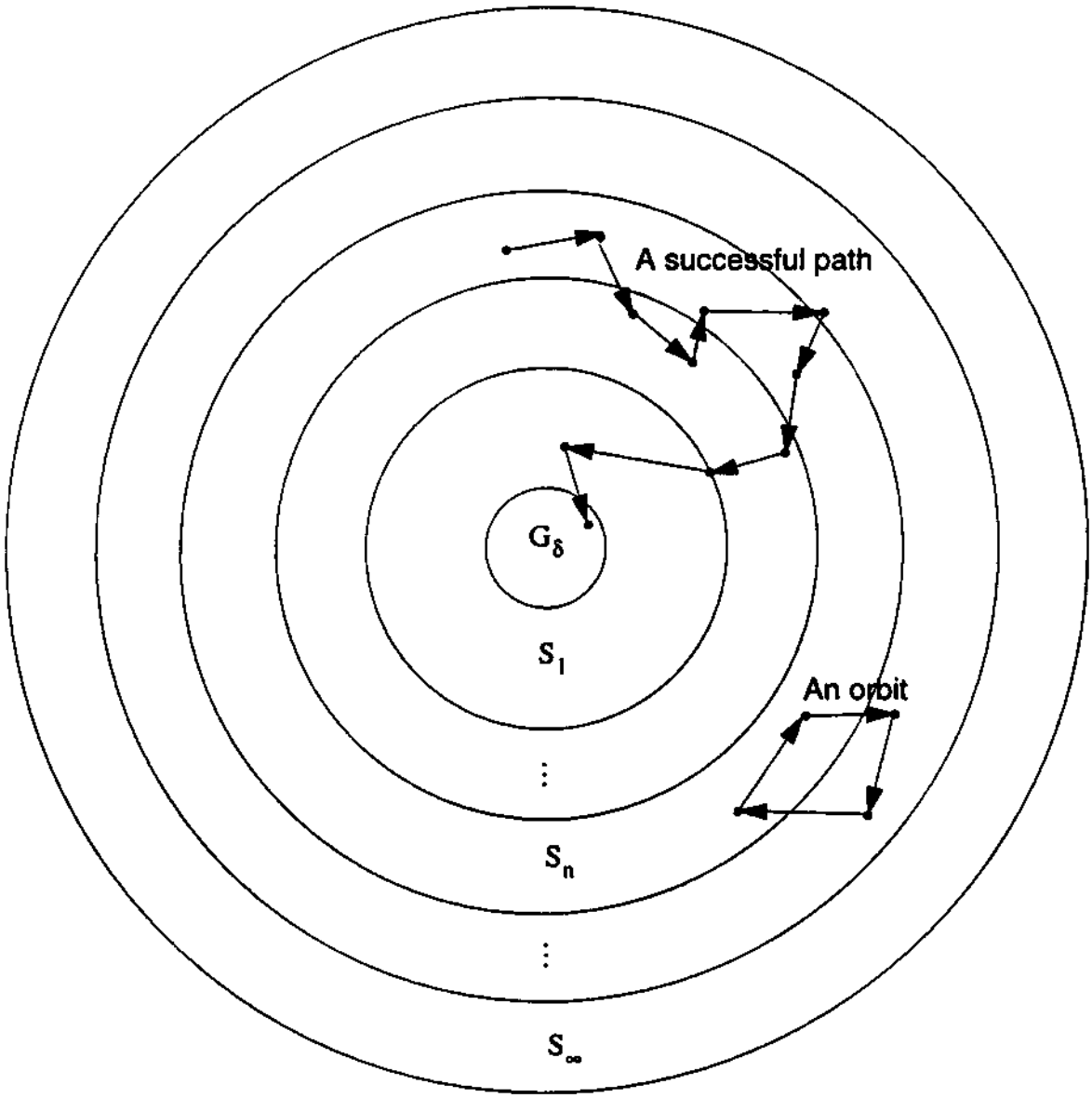

Fig 6. The space of solutions is partitioned into regions so that all the points in S_n are n-constructive operations from the goal space G_5. These regions, particularly the outer ones, depend on both G5 and C. As G5 contracts, the outer regions expand; as C expands, the outer regions contract.

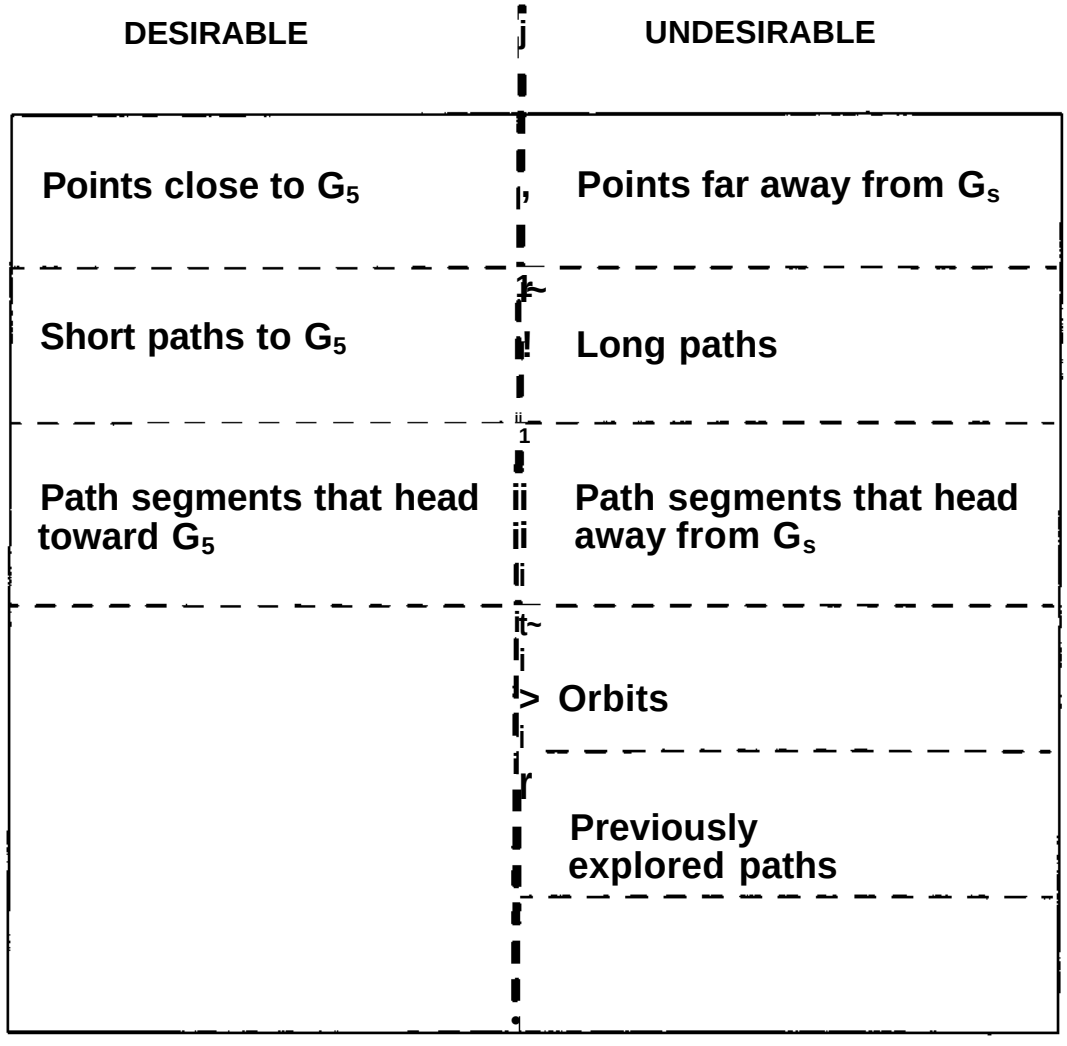

Fig 7: H, the set of all possible paths in S_f can, in principle, be partitioned into desirable paths and undesirable paths. The function of the destroyers is to recognize and erase undesirable paths before the constructors have wasted a great deal of time on their development.

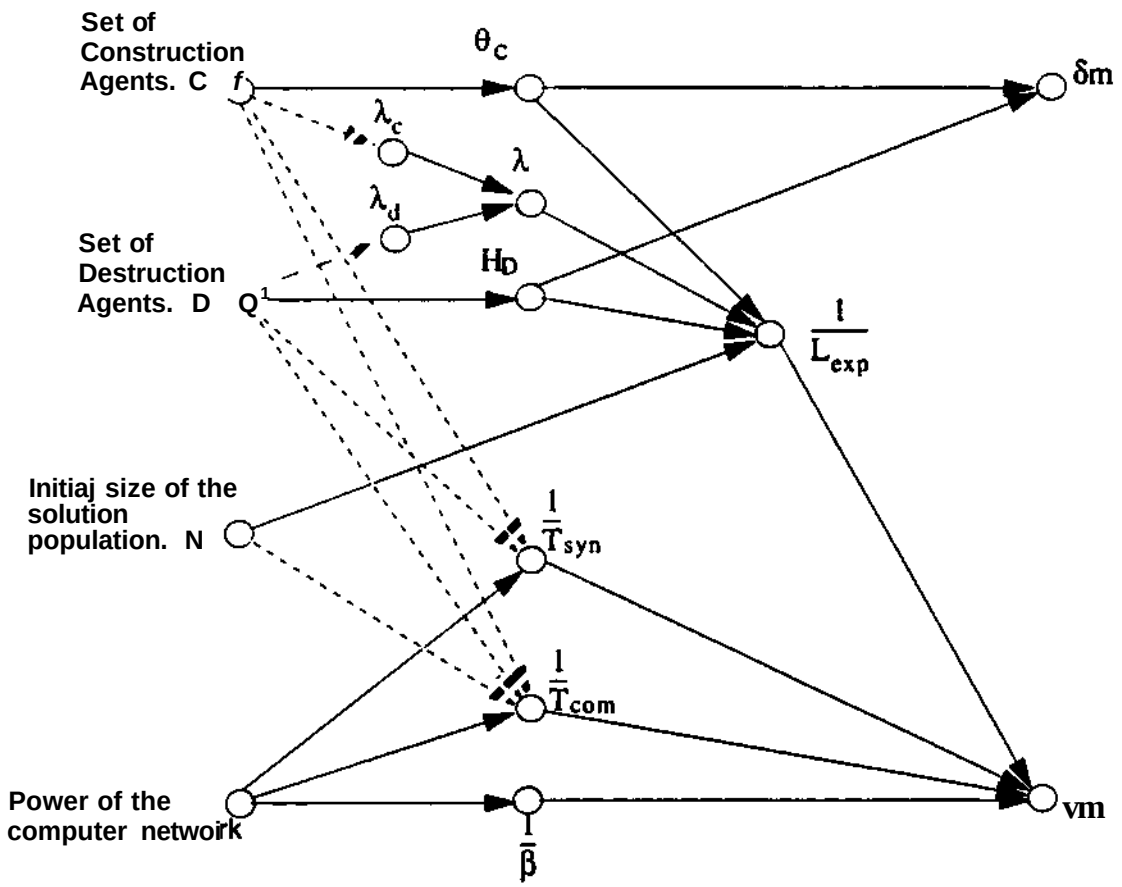

Fig. 8: Causal relations of a CDM. Each solid arc denotes a monotonically-increasing relationship. For instance the solid arc between N and $\frac{1}{L_{exp}}$ means that $\frac{1}{L_{exp}}$ increases monotonically with N.

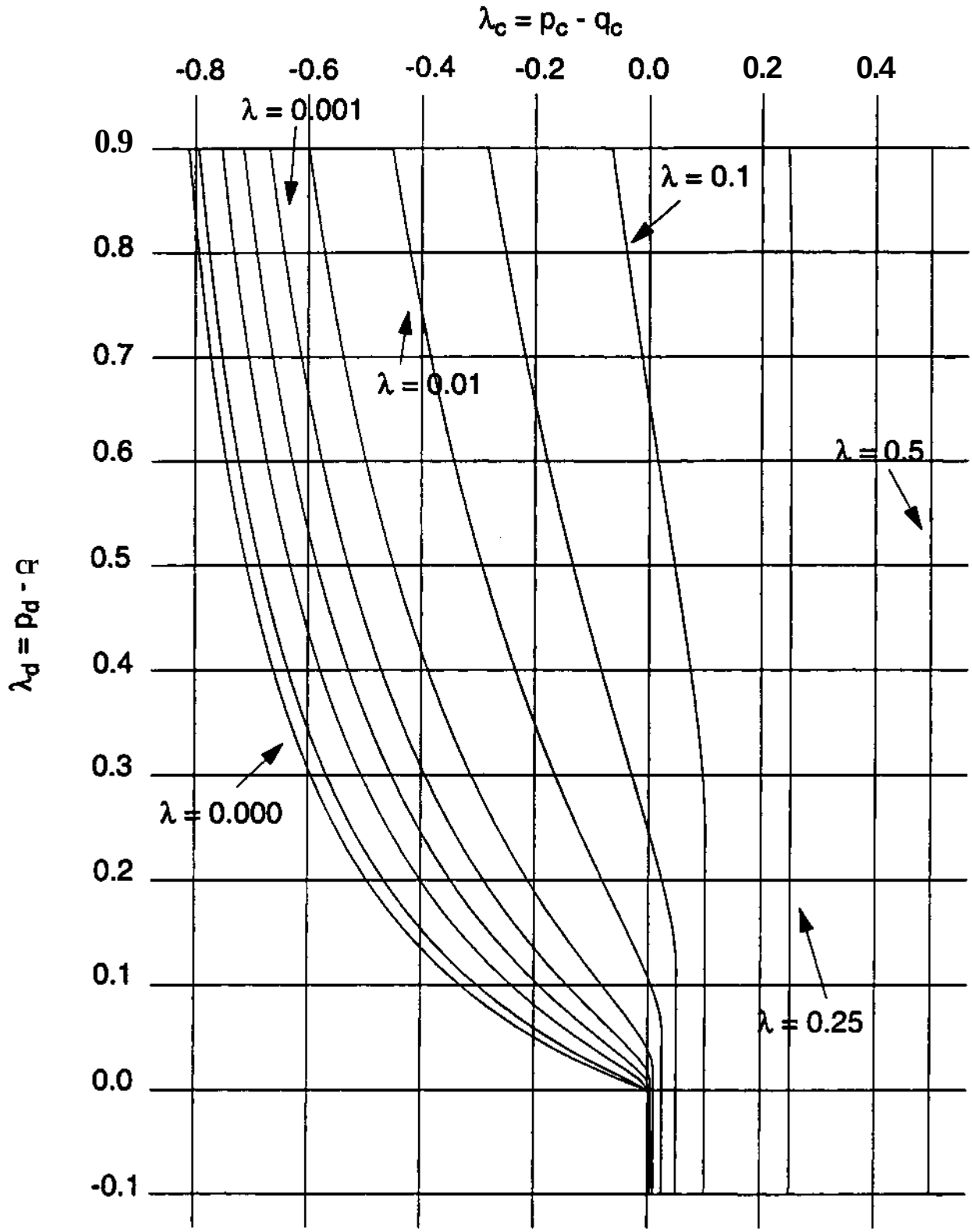

Fig. 9: Iso-drift curves for one set of values of q_d, r_d and r_c. Notice how *X*, the overall drift depends on *XQ*, the construction drift, and *X^*, the destruction drift

APPENDIX

In this appendix we explore CDMs in greater detail.

Definitions:

Let:

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

0_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.

5 be an indicator of solution-quality such that 8 increases as solution-quality increases.

G_5 be the subset of S that contains all the solutions of quality 5 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_8 be the expected amount of time for the population of solutions to evolve at least one solution of quality 5 or better. G_5 is said to be reachable if $T_§$ is finite.

5m be the greatest value of 5 such that G5 is reachable,

vm = 1/Tgm be the expected speed with which G^ is reached.

d(y) be the distance of y from G5, 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 G5.

S_n be the subset of S containing all the solutions that are at a distance of n from G5, 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_6, a regressive edge moves it further away and a neutral edge leaves it at the same distance.

Pc Qc and rc be the values of P, 3 and r when the destroyers are disabled.

Pdk« 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.

X = p-q be the overall drift of the CDM; $X_c = p_c - q_c$, be the drift of the constructors; and *X*^ = $p_d - q_d$ be the drift of the destroyers.

p 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_{8m}).

Calculating the overall drift *X* 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. *X* 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

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

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

Reachability of G_s

If G5 is small or the set of construction agents is weak, not all solutions in S are at a finite distance from G5. Let S^ be this set of points for which there is no path to G5. Clearly we cannot guarantee reaching G§ unless the destroyers make S^ inaccessible.

If:

- *X* 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, $S_k = 0$ or $S_k \subset H_D$

then:

- G_s is reachable.

proof:

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

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

so when $X > 0$ we get a finite j. Starting with a solution in S_n the expected number of steps required to get to G5 is ^. let E[n] denote the expected distance to G5 from the randomly seeded solution. Since all solutions not made inaccessible by the destroyers are at distance at most K we have E[n] < «>. Hence then expected time to goal from the randomly seeded solution is $\frac{E[n]}{K} <$ <».

Monotonic relationships. (Fig. 7)

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 G5. 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_s) of these points is identified and a path from it to G_6 is developed by the sequential application of construction agents and destroyers;

Then:

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

Ri» ^2» ^3» — decrease monotonically as the variety of constructive skills in increases, that is, as the number of agents in C increases (2)

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

$$L_{exp} = X^{n}min \quad (4)$$

where n_{min} is the expected distance of the best starting point from G_s, and L_{exp} is the expected length of the path from this point to G5.

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 6_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

$$n_{min} = \sum_{n=0}^{\infty} n \cdot R - R_{n\wedge;} = > nR_n - > (n-1)R_n = > R_n$$

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

Now when N increases each term in the sum (3) decreases so the expected speed $v_m = \frac{L}{n_{min}}$ with which we reach G5 increases.

Likewise by (2),(4) an increase in 8_C or *X* also causes the v_m to increase.

2.Solution-quality,8_m.

Soo depends on 6 and 6_C. Specifically S^ fills as 8 increases, and empties as 6_C expands. Suppose that an improvement of solution quality from 8 to 8-A8 causes S^ to fill by the amount AS^,. Then sufficient conditions for achieving this improvement are: an expansion of 0_C to empty part of AS^, and an expansion of H_D to contain the rest of AS^, all while maintaining $X > 0$.

Age Based Restarting.

For small A„ 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 $n° = (\text{FIQ}, \text{rip} \ldots J$ be the distribution for the best member in the set of initial solutions, that is $U°_n$ = prob[best initial solution e S J , 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 $n^N = n°P^N$ 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

$$Q_n^N = \Pr\left[\text{sol in } S_n(9, 8) \text{ after N iterations) sol not in } G_{\S} \text{ after N iterations}\right] = \frac{n^N}{1 - \pi^{\wedge} - \wedge}$$

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_6 is then

$\sum^{oo} \frac{n}{pqq} Q_n^N$ For certain values of p, q, r and n°, 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 n° 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_5 is

$$N(\text{ exp. \# of restarts }) + (\ ^{ex}\text{P- \# °f iterations to reach } G_5 \text{ reached in <-N iterations } J$$

$$= N\left(\frac{1-\Pi_0^N}{\Pi_0^N}\right) + \sum_{t=1}^{N} t P r (X \ll -1 \ll G 5^{AX} . e^{G} 8)$$

$$\left(1-\Pi_{\cdot}^{N}\right)$$

As a numerical example take p = 0.52, r = 0, q = 0.48 and assume that the best newly created solution will always start in S_{10}. Then the expected number of iterations to get to Gg without age based restarting is 250. With age based restarting at N = 110 we get UQ^{N} = 0.48862 and expected number of iterations to G_5 equal to 171.202 < 250.

Note that the restart threshold N that minimizes the expected time to reach G§ is not necessarily the first N for which the expected number of remaining iterations exceed the expected number of iterations to $G_{\S}$ from the best initial solution. If the variance on the expected remaining iterations is sufficiently high, and it normally is, it is more advantageous to do a few extra iterations before restarting. The best value for N is found by calculating (*) for a different values of N. It can be shown that (*) is concave in N for fixed p, q, rand n^{u}.