A-Teams: Multi-Agent Organizations for Distributed Iteration

S.N. Talukdar, P.S. deSouza, R. Quadrel, V.C. Ramesh

EDRC 18-30-92

A-TEAMS: MULTI-AGENT ORGANIZATIONS FOR DISTRIBUTED ITERATION

S. N. Talukdar

P. S. deSouza

R. Quadrel

V. C. Ramesh

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA., 15213
(e-mail: snt@edrc.cmu.edu)

Theme: Coordination as Search

Proceedings of the EPRI/NSF Workshop on Application of Advanced Mathematics to Power Systems, Redwood City, CA Sept 3 - 6, 1991.

A-TEAMS: MULTI-AGENT ORGANIZATIONS FOR DISTRIBUTED ITERATION

S. N. Talukdar, P. S. de Souza, R. Quadrel, V. C. Ramesh
Engineering Design Research Center
CMU, Pittsburgh, PA., 15213

ABSTRACT

An A-Team is a multi-agent organization for cyclic (iterative) processing. All the agents are autonomous. The cycles are asynchronously coupled, allowing them to be run in parallel and to collaborate without delaying one another. We conjecture that there are simple and effective ways of designing these couplings and offer three examples in support of the conjecture: A-Teams for solving sets of nonlinear algebraic equations, travelling salesman problems, and high-rise building design.

INTRODUCTION

A multi-agent system can be decomposed into two parts: (a) an organization consisting of the agents and mechanisms for their interaction, and (b) a computing environment for building and using the organization. Here we will focus on the organization, beginning with some terminology, then defining A-Teams, and finally, presenting some examples of the design of A-Teams.

TERMINOLOGY AND MODELS

Every multi-agent organization must contain a set of stores for the data its agents use and produce. This section develops models of the relations among agents and stores from three viewpoints: topology, operating policy and openness (potential for growth).

TOPOLOGY

Harel has developed a formalism called a Higraph [Harel 88] that is convenient for visualizing most features of organizational topology. In the succeeding material we will modify Higraphs by the addition of a few additional features to obtain what we call a Tao graph. We will also develop an algebraic equivalent of a Tao graph that is useful in analysis.

Aspects and Stores

Complex problem-solving processes invariably involve large amounts of data and many different representations. In recognition of this fact, we will use a coarse unit of data called an *aspect*.. Conceptually, an aspect is a view, model or partial description of some object of interest. For example, circuit diagrams, lists of materials, operating manuals and behavioral specifications are some of the many aspects of an electric motor. More formally, an aspect x is a double: x = (R, V), where R is the representational scheme used by the aspect and V is a collection of values that instantiates this representation.

We define a store as a set of aspects. A store is said to be *homogeneous* if all its aspects use the same representational scheme, otherwise, it is *heterogeneous*. Stores and their Cartesian products are denoted by closed figures (Fig. 1).

Agents and Tasks

We define an agent as an operator capable of mapping the elements of one store into those of another. This mapping may be one-to-one, one-to-many or many-to-many, it may be deterministic or stochastic, and it may be into or onto. Pictorially, we will denote all these types of mappings by directed arcs as shown in Fig. 2. Symbolically, an agent is denoted by the relation:

$$y = f^{k}_{ij}(x),\ x \in S_i \ \text{ and } y \in S_j \qquad \text{(E1)}$$

where k is the agent's name, S_i is the agent's input store and S_j is its the output store.

Any computational task can be thought of as a transformation of a given aspect into a goal aspect. The task of designing a motor, for example, is equivalent to transforming an aspect that specifies the motor's behavior into an aspect that describes its structure. Thus, a task is defined by two aspects; the first is given; the second is to be calculated. If the semantic gap between these aspects is too large to conveniently bridge with a single agent, then intermediate stores may be placed to serve as stepping stones along the way. The connecting sequence of agents and intermediate stores is called a *computational path* or *string.*

Tao Graphs, Data Flows and Authority Flows

The purpose of a Tao graph is to help visualize the topology of an organization--the relative locations of its agents and stores, the tasks the organization can perform, the different paths it may take in performing these tasks, and the supervisory relations among its agents.

A Tao graph has two components: a data flow and an authority flow.

An *authority flow* is a set of broken arrows that represent the supervisory relations among agents. As such, an authority flow is equivalent to the"organization chart" that is traditionally used to depict who supervises whom in human organizations. The four principal types of authority flows are: a simple hierarchy (Fig. 3), a compound hierarchy (one with more than two levels), a matrix (at least one agent has more than one supervisor), and a null flow (there are no supervisors and all the agents must be completely autonomous).

A *data flow* is a directed graph whose nodes represent stores and whose arcs represent agents. We say that a data flow is *functional redundant* if at least one of its stores can be reached by more than one path. Cycles in a data flow (Fig. 4) make feedback and iteration possible, which in turn, make possible the reactive improvement and correction of computed results.

Note that all the information in a data flow can be expressed by a set of equations obtained by writing (E1) for every agent in the organization, as shown below:

$$y = f^k_{ij}(x), \text{ for } x \in S_i,\ y \in S_j, \text{ and } \forall\, k \in K \qquad \text{(E2)}$$

where K is the set of all agents. Though visually less appealing than a data flow, this set of equations has uses that we will get to shortly.

OPERATING POLICY

The operating policy of an organization is a collection of rules and regulations that govern its temporal activities. Two of the main issues are (c.f. (E2)):

• *coordination*: how is the input, x, for each agent to be selected from among the many entries that could accumulate in its input store? Does this selection require global information or can it be made locally?

• *coupling*: do the interactions among agents impose an order on their invocation? Can some or all of the agents work in parallel?

The coordination policy has a large effect on the quality, convergence and stability of an organization's computations. In addition, it influences the coupling among the agents. To illustrate, consider the following coordination policies for iteration with the cyclic data flow of Fig. 4:

(OP1): $$v_{n+1} = g(v_n, w_n) \qquad n = 0,1,2,\text{-----}$$
$$w_{n+1} = h(v_{n+1}, w_n)$$

(OP2): $$v_{n+1} = g(v_n, w_n)$$

$$w_{n+1} = h(v_n, w_n) \qquad n = 0,1,2,\text{-----}$$

(OP3): $$v_{n+1} = g(v_n, w_*) \qquad n = 0,1,2,\text{-----}$$
$$w_{n+1} = h(v_*, w_n)$$

where "*" means "latest available," n is the iteration count, and $v \in V$, $w \in W$.

The choice of inputs dictated by the coordination policy of (OP1) requires the agents h and g to be operated serially in alternating order: g, h, g, h,---. The coordination policy of (OP2) is less restrictive. It allows the agents to be operated in parallel, provided the faster agent waits for the slower one between iterations. Thus, it allows the sequence: g//h, g//h,----. The coordination policy of (OP3) places no restrictions on the order in which the agents may be invoked; both agents may proceed in parallel and at different speeds.

When no agent has to wait for any other, we will say that the agents are *asynchronously* coupled. The advantage of asynchronous coupling is that it allows all the agents to work in parallel all the time. The disadvantage is that it is more prone to divergence and instability.

GROWTH

Two attributes that are useful in characterizing an organization's potential for growth are:
•*unit-of-growth*: the quantum of expansion. Typical quanta are agents and stores.
•*cost-of-growth:* the cost of any modifications that must be made to the organization to make it capable of accepting and using a new unit. Note: this cost does not include any expenses incurred in assembling and packaging the unit-of-growth.

A-TEAMS

Rationale

An A-Team (Asynchronous Team) is an organization-type that has evolved from a traditional blackboard [Nii 86a, b]. This sort of blackboard (Fig. 5) has three weaknesses. First, it does not allow its agents to work in parallel. Second, the single, centralized store is difficult to expand. Third, the system is overly dependent on its supervisor; errors made by the supervisor critically affect performance; also the supervisor must be modified every time a new type of agent is added. The first two weaknesses are easily remedied. One could, for example, assemble a number of blackboards, distribute them over a network of computers, and assign some agents from each blackboard to handle interactions with other blackboards [Leão 88]. One approach to eliminating the third weakness would be to seek more dependable and self modifying supervisors. Instead, we have chosen to eliminate the need for supervision.

Definition

We define an A-Team as any organization whose authority flow is null (there are no supervisors), whose data flow is cyclic, and whose agents are asynchronously coupled (so all the agents can work in parallel virtually all the time).

A general form of the topology of an A-Team (Fig. 6) is very like that of its ancestor, the traditional blackboard (Fig. 5). However, there are differences in their operation and growth. Since A-Teams are often confused with blackboards, it is worth wile to point out some of these differences as is done in Table 1. Clearly, a traditional blackboard is not an A-Team. However, one can make a strong case for including both scientific communities, as described in [Kornfeld 81], and insect societies in the class of A-Teams. The arguments are given in [Talukdar 91]; we do not have the space to reproduce them here. We merely note that any organization which can serve the needs of agents as diverse as scientists and insects must have many strengths. These include easy growth (new agents and stores can be added without making any modifications to the system), high dependability (high functional redundancy allows some agents or computational paths to fail without compromising overall performance), and high performance (large numbers of agents working in parallel allow wide spaces to be searched in relatively short times).

Attribute	**A-Team**	**Traditional Blackboard**
Topology		
Authority flow	none	simple hierarchy
Data flow	strongly cyclic	weakly cyclic
Functional redundancy	usually high	low to moderate
Operating Policy		
Coordination	local, often randomized	local
Coupling	asynchronous	synchronous
Growth		
Unit-of-growth	string	agent
Cost-of-growth	none	moderate to high

Table 1. Signatures (the values of some important organizational attributes) of an A-Team and a traditional blackboard.

Designing A-Teams

There are two principal steps to designing an A-Team:
• select a data flow; and
• devise a coordination policy for its agents. This policy must be locally implementable and allow the agents to be asynchronously coupled. (There are no supervisors to enforce coordination policy. Instead, each agent must make its own decisions with information contained in its input store.)

A Conjecture

We conjecture that there are a number of problem domains for which appropriate coordination policies (that is, policies that are local and asynchronous) can be devised that are simple (requiring no more than a few rules to be added to each agent), and effective (resulting in teams with high performance)

Some support for this conjecture is provided by the following observations.
• Consider a cyclic data flow with no functional redundancy (Fig. 4, for example), and the coordination policy of (OP3), namely: select the latest aspect placed in the input store. This policy is simple and local. Moreover, sufficient conditions for its convergence are only slightly more restrictive than the sufficient conditions for any synchronous policy, such as (OP1) [Talukdar 83].
• Many biological organizations with autonomous agents and a great deal of functional redundancy, flocks of birds and schools of fish, for example, appear to use simple, local and asynchronous coordination policies to produce collective behaviors that are quite complex [Ermentrout 91],[Reynolds 87], [Heppner 90].

In the next section we will provide slightly stronger support for our conjecture--some preliminary results from three computer-based A-teams that we have constructed.

MULTI-ALGORITHM PROBLEMS

A multi-algorithm problem is one for which: (a) several iterative algorithms are available, and (b) all the available algorithms are prone to failure. Sets of nonlinear algebraic equations and NP-hard problems, such as the travelling salesman problem (TSP), are two examples. We will attempt to design A-Teams for these examples that combine algorithms so as to capitalize on their strengths and protect against their failures.

SETS OF NONLINEAR ALGEBRAIC EQUATIONS

The Problem

Given a set of nonlinear algebraic equations: $F(X)=0$, the problem is to find a solution, X_*, such that $\|F(X_*)\|=0$. . There are two types of iterative algorithms for searching for X_*. The first has the form: $X_{n+1}=\vartheta(X_n)$, $n=0, 1, ---$, where X_n is an approximation to the solution, ϑ is an algorithm-specific function, and decreasing values of $\|F(X_n)\|$ indicate that the algorithm is making progress. The Levenberg-Marquardt (LM) algorithm is widely held to be one of the best of this type. Other examples are Newton-Raphson and Gauss-Seidel

The second type of algorithms has the same form as the first but uses populations (sets) of approximations in place of individual approximations. A genetic algorithm (GA) is an example, and has the form: $X_{n+1}=\Psi(X_n)$, $n=0, 1, ---$, where X_n is a population of approximations.

An A-Team Design

The data flow of an A-Team that combines algorithms of both types is shown in Fig. 7. The coordination policy for each LM-based agent is as follows: continue with (A1) as long as progress is being made, otherwise replace X_n with X_*, the best solution available from the store. The genetic algorithm is run continuously from a randomly selected starting population. In effect, the GA, which is good at global searches, is used to bring the LM algorithms into their effective range from a solution. In this range they are very fast and reliable. Outside it, they are tentative and slow.

Results

The coordination scheme ensures that the performance of the team as a whole will be no worse than that of its strongest member, working alone. In the tests conducted so far, the team has done much better than this lower bound, usually requiring many less function evaluations than either a genetic or LM algorithm, and finding solutions for problems where these algorithms fail [de Souza 91].

THE EUCLIDEAN TSP (TRAVELLING SALESMAN PROBLEM)

The Problem

The Euclidean TSP is to find a tour (circuit) of minimum Euclidean length that connects each of a set of cities with given positions. The available algorithms, and there are many of them, fall into two classes: construction algorithms and improvement algorithms. The former transform a partial tour into a complete tour; the latter, improve complete tours.

An A-Team Design

The data flow for an A-Team that combines several of the simpler construction and improvement algorithms is shown in Fig. 8. The coordination policy used is very simple. All agents except D (the destroyer), select their inputs randomly and without repetition from their input stores. D limits the contents in the complete-tour-store to the best N tours, N being a user selected parameter. Thus, D provides a "survival-of-the-fittest" screen that ensures monotonic improvement of the contents of the complete-tour-store. The process terminates when these contents reach a steady state. Many such states are possible, not all of them optimum. The randomization causes the terminal steady state to vary from one run to another.

Results

We have tested the team on a number of problems including two of the larger ones for which optimum solutions are known: the 318-city problem of Lin and Kernighan [Lin 73] for which the optimum tour is of length 4,1345 [Crowder 80], and the 532-city problem of Padberg and Rinald for which the optimum tour is of length 27,686 [Padberg 87]. In all the tests, the team performs much better than its members can. For instance, we made four runs on the 318-city problem. One found the optimum and the other three came within 0.01% of the optimum. In contrast, the best that any of the algorithms can do when working alone, even from exceedingly favorable starting conditions, is to come within 5% of the optimum (and this computation took far longer than was required by the team).

We have also made four runs with the 532-city, problem obtaining solutions of 27,703 (0.061% above optimum), 27,704 and 27,705 twice. In contrast, Johnson reports that in 20,000 independent runs of the LK (Lin-Kernighan) algorithm the best result obtained was 27,705 [Johnson 90]. Note that the LK algorithm is one of the best in existence, and is far superior to any that we have, as yet, been able to code or obtain for our team. We expect the performance of our team to improve with the addition of better algorithms.

CONFLICT RESOLUTION IN £ + N DESIGN

The Problem

Design problems often involve many more criteria (objectives and constraints) than existing synthesis procedures can handle. The design of high rise buildings is a case in point. Some of the many classes of criteria: are client-specifications, building codes, initial cost, aesthetics, functionality, comfort, maintenance, construction, and eventual demolition. Existing design processes can address only a very small fraction, say 6, of these criteria. We call such processes *£ design.* One way to increase the coverage of an £ design process is to add feedback loops for other criteria. For example: produce an initial design that takes specifications, codes and cost into account; then evaluate this design from the viewpoints of thermal performance and construction; and finally, modify the design if it is inadequate from these viewpoints. The modification (feedback) signals can be generated by expert systems that are beginning to be called *critics* or *design advisors.* One difficulty is that the signals they produce are often in conflict. Lacking mechanisms to resolve these conflicts, many researchers have resigned themselves to using only one advisor at a time. We **call** this £ +1 design. The problem that we consider here is to incorporate N > 1 feedback loops into a design process along with conflict resolution mechanisms that drive solutions to Pareto optimality. We call such a process £ + *N design* to signify that e criteria are handled by feedforward and **N** criteria by feedback.

An A-Team Design

Fig. 9 shows the data flow of an A-Team for £ + 3 design. This team was formed by adding three evaluators: E1, E2, E3, three design advisors: a, {5, y, and a destroyer: D, to an existing system for £ design of high-rise buildings.

The coordination policy is similar to that of the team for TSP; all agents except D select their inputs randomly and without repetition from their input stores. D repeatedly checks all the designs **in its** store. Any that is found to be inferior to the others (in the Pareto sense with respect to the evaluation criteria: C, T and L) becomes a candidate for elimination. Whether it is actually eliminated **or** not is determined by a random process like that used in simulated annealing; the chances of **elimi-** *
nation increase with the candidate's inferiority. The effect of D is to allow a few inferior designs **to** remain in a population most of whose members are moving toward the Pareto frontier. As **in simulated** annealing, these inferior candidates provide some of the diversity needed to obtain **good** coverage of the search space. Further details can be found in |Quadrel 91|.

Results

The original £ design process is deterministic and generates one design per run. The A-**Team is** non-deterministic and produces many designs per run from which the user may select the one **he or** she likes the best. So far, we have conducted only one test on the A-Team. It produced **five de** signs, all of them superior in all respects (constructability, thermal performance and lighting) to the original design [Quadrel 91].

FUTURE WORK

The work reported here indicates that A-Teams have promise. In our continuing work, we are are expanding the teams already built as well as moving to two new domains: real-time control of electric power systems and network diagnosis.

REFERENCES

[Aho 83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, "Data Structures and Algorithms", Addison-Wesley, Reading, MA, 1983.

[Crowder 80] H. Crowder and M.W. Padberg. Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality. *Management Science,* Vol. 26, No. 5, May 1980.

[Decker 87] Keith S. Decker. Distributed Problem-Solving Techniques: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics.* September/October, 1987.

[de Souza 91] 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, July 1991.

[Erman 80] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy. The Hearsay-H Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys.* ACM. Vol. 12, pp. 213-253, June 1980.

[Ermentrout 91] B. Ermentxout, "An Adaptive Model for Synchrony in die Firefly Pteroptyx Malaccae", *Journal of Mathematical Biology,* Springer-Verlag, Vol 29, pg. 571-585.

[Fletcher 87] R. Fletcher. "Practical methods of Optimization", Chirchester, UK: John Wiley & Sons, 1987.

[Fox 88] Mark Fox. An Organizational View of Distributed Systems. AJL Bond &L. Gasser (eds.), *Readings in Distributed Artificial Intelligence,* pages 140-150. Morgan Kaufmann Publishers, Inc., 1988.

[Garey 79] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to NP-completeness", WJH. Freeman, San Francisco, CA, 1979.

[Goldbergh 89] D.E. Goldebergh. "Genetic Algorithms in Search, Optimization & Machine Learning", Reading, MA: Addison-Wesley, 1989.

[Goldbergh 87] D.E. Goldebergh and J. Richardson. "Genetic Algorithms with Sharing for Multimodal Function Optimization", Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[Harel 88] D. Haiel. On Visual Formalisms. *Communications of the ACM.* Vol. 31, No. 5, May 1988.

[Heppner 90] Frank Heppner and U. Grenander, "A Stochastic Nonlinear Model for Coordinated Bird Flocks", from The Ubiquity of Chaos, ed. Saul Krasner, Washington: AAAS Publ. 1990.

[Hewitt 88] C Hewitt Officer are Open Systems. *Readings in Distributed Artificial Intelligence.* A.H. Bond and L. Gasser (eds.), Morgan Kaufmann, 1988.

[Johnson 90] D.S. Johnson. Local Optimization and the Travelling Salesman Problem. *Lectures Notes in Computer Science,* G. Goos and J. Hartmanis (eds.), Automata, Languages and Programming, 17th Int. Coll.,. Worwick University, Eng., July 90, Springer-Verlag, Vol. 443.

[Kornfeld81] W.A. Kornfeld and C.E. Hewitt. The Scientific Community Metaphor. IEEE Trans. Sys., Man, Cybern., Vol. SMC-11, pp. 24-33, Jan. 1981.

[Leão 88]	Leão, L.V. and Talukdar, S.N., "COPS: A System for Constructing Multiple Blackboards," Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers, Inc., Les Gasser and Alan Bond (eds.), San Mateo, CA, 1988.
[Lin 73]	S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, Vol. 21 (1973), pp. 498-516.
[Mullender 89]	S. Mullender. Distributed Systems. *Addison-Wesley*. 1989.
[Nii 86a]	H. Penny Nii. Blackboard Systems: The Blackboard Model of Problem Solving
[Nii 86b]	H. Penny Nii. Blackboard Systems: The Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective, *The AI Magazine*. August 1986.
[Padberg 87]	M. Padberg and G. Rinald. Optimization of a 532-city Symmetric Travelling Salesman Problem by Branch and Cut. *Operations Res. Lett.*, Vol. 6 (1987), pp. 1-7.
[Quadrel 91]	Richard Quadrel. Asynchronous Design Environments: Architecture & Behavior. PhD Thesis. Department of Architecture. Carnegie Mellon University. September 1991.
[Reynolds 87]	Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model", *Computer Graphics*, Vol. 21, No. 4, p. 25-33, July 1987.
[Rychener 88]	M.D. Rychener. Expert Systems for Engineering Design. *Academic Press*. 1988.
[Simon 76]	Herbert Simon. The Design of Large Computing Systems as an Organizational Problem. In P. Verberg et al, Organisatiewetenschap en praktijk, 1976.
[Talukdar 83]	S.N. Talukdar, S.S. Pyo and R. Mehrotra, "Distributed Processors for Numerically Intense Problems", Final Report for EPRI Project, RP 1764-3, March 1983.
[Talukdar 89]	S.N. Talukdar and S. Fenves. Towards a Framework for Concurrent Design. *Proceedings of the MIT-JSME Workshop on Cooperative Product Development*. D. Sriram. R. Logcher, and S. Fukuda (eds.), Cambridge, MA: MIT, Nov. 1989.
[Wilson 71]	E.O. Wilson. *The Insect Societies*. Belknap/Harvard Press, 1971.

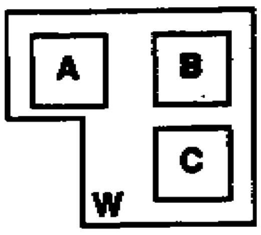

Fig 1. Four stores such that: W = A x B x C

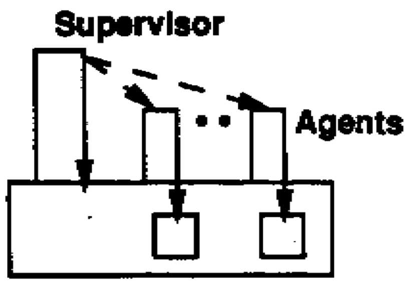

Fig 5. The topology of a traditional blackboard. The agents work serially in an order determined by the supervisor.

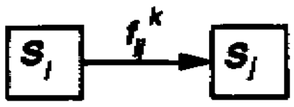

Fig 2. An agent that maps from S_i to S_j.

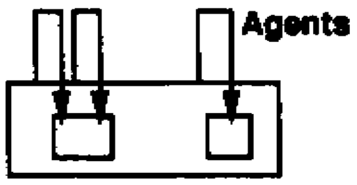

Fig 6. The general form of an A-Team's data flow.

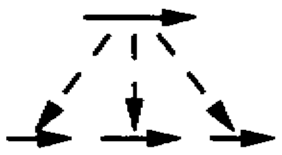

Fig 3. A simple hierarchy.

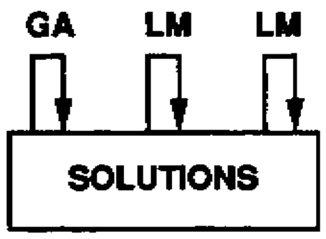

Fig 7. Data flow of an A-Team for solving sets of nonlinear algebraic equations. GA: genetic algorithm. LM: Levenberg-Marquardt algorithm.

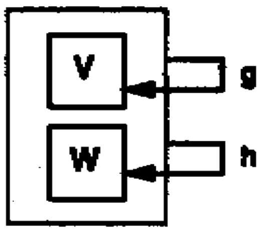

Fig 4. Cycles in a data flow make iteration possible.

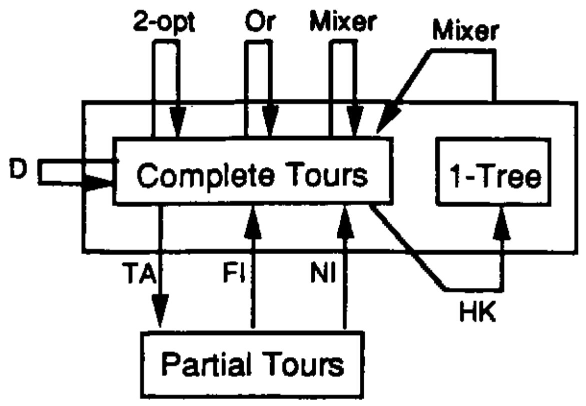

Fig 8. An A-Team for the travelling salesman problem. FI (Furtherest Insertion) and NI (Nearest Insertion) are constructive algorithms. 2-Opt and Or are improvement algorithms. The HK (Held-Karp) algorithm calculates a lower bound on the optimal tour length and also near-optimal but infeasible solutions called 1-Trees. The TA (Tour Analyzer) creates partial tours consisting of the common edges from two randomly selected complete tours. The Mixers randomly mix edges from two tours to create one tour. D (Destroyer) eleminates all but the best N tours from the Complete Tour store. (N is a user selected parameter.) All agents except D select their inputs randomly from among those available in their input stores.

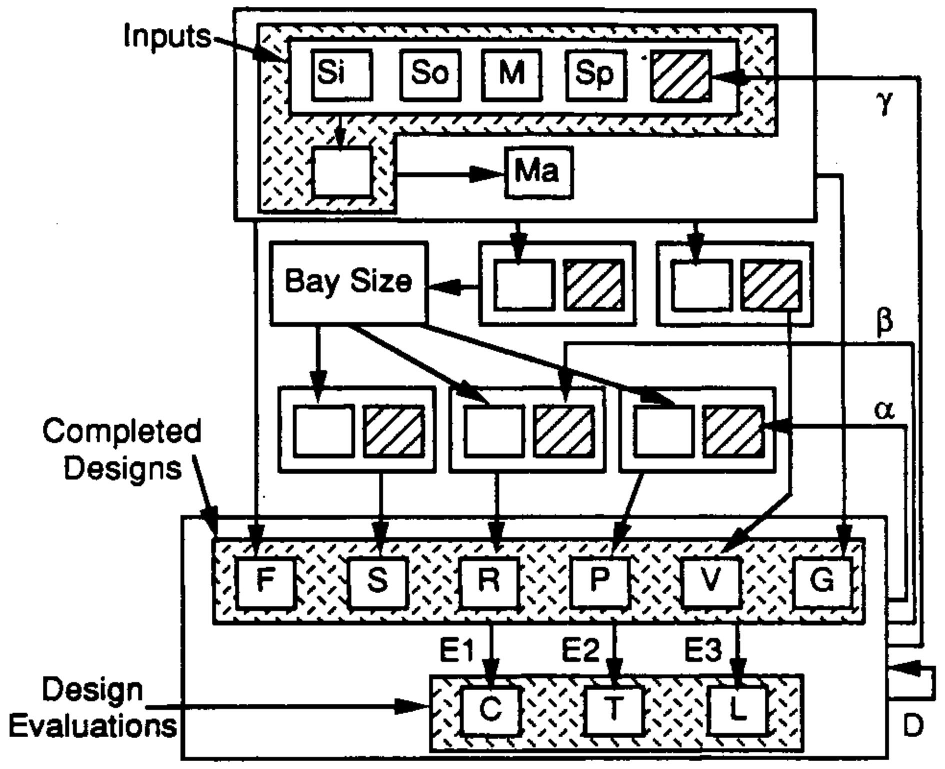

Fig 9. The data flow of an A-team for the design of high-rise buildings. Only the more important stores and agents are labeled. Si: Site Data; So: Soil Data; M: Materials; Sp: Customer Specifications; Ma: Massing; F: Foundation; S: Structural System; R: Roof System; P: Panels; G: Garage; V: Vertical Circulation; C: Constructability; T: Thermal Performance; L: Lighting; ▨ : Design Constraints; E1, E2 and E3: Evaluators; α, β, γ : Design advisors; D: destroyer.