

# **A-Teams: An Agent Architecture for Optimization and Decision-Support**

**John Rachlin<sup>1</sup>, Richard Goodwin<sup>1</sup>, Sesh Murthy<sup>1</sup>, Rama Akkiraju<sup>1</sup>,  
Fred Wu<sup>1</sup>, Santhosh Kumaran<sup>2</sup>, Raja Das<sup>2</sup>**

<sup>1</sup> IBM T.J. Watson Research Center  
Yorktown Heights, NY

<sup>2</sup> IBM Supply Chain Optimization Solutions  
Atlanta, GA

## **Abstract**

The effectiveness of an agent architecture is measured by its successful application to real problems. In this paper, we describe an agent architecture, A-Teams, that we have successfully used to develop real-world optimization and decision support applications. In an A-Team, an asynchronous team of agents shares a population of solutions and evolves an optimized set of solutions. Each agent embodies its own algorithm for creating, improving or eliminating a solution. Through sharing of the population of solutions, cooperative behavior between agents emerges and tends to result in better solutions than any one agent could produce. Since agents in an A-Team are autonomous and asynchronous, the architecture is both scalable and robust. In order to make the architecture easier to use and more widely available, we have developed an A-Team class library that provides a foundation for creating A-Team based decision-support systems.

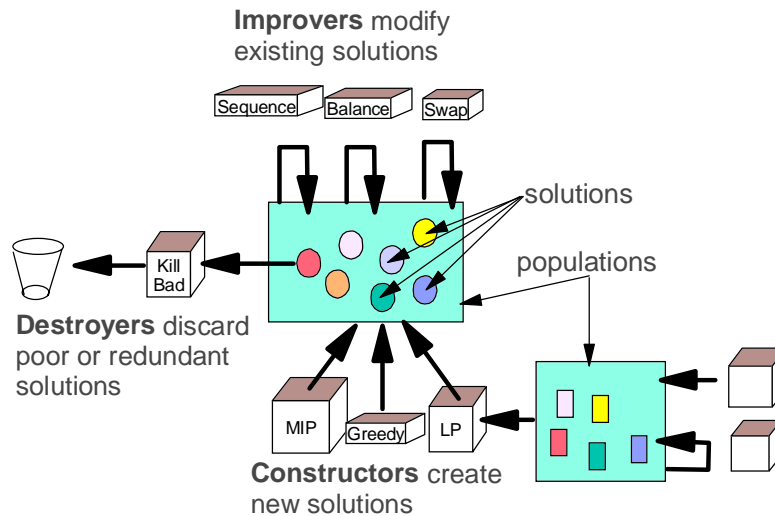
## **1. Introduction**

This paper describes an implementation of A-Teams (Talukdar et al, 1983, Talukdar et al, 1993; Talukdar et al, 1996), a multi-agent architecture which we have used to successfully deploy multi-objective decision-support systems for complex, real-world domains. Problem solving in these domains requires searching a large multi-dimensional space to find efficient solutions. The A-Team framework enables us to easily combine disparate problem solving strategies, each in the form of an agent, and enables these agents to cooperate to improve the quality and diversity of the resulting solutions. Humans, acting as agents, can use their domain expertise to guide the search and to expand the set of possible solutions by relaxing constraints. By combining the speed and accuracy of software agents with the deep knowledge of human experts, A-Team based decision-support systems improve the decision making process, which has led to significant economic advantages for our customers (Shaw 1998, Hoffman 1996).

We begin our discussion with an overview of the A-Team architecture and describe some of its advantages for creating decision-support systems. We then discuss related work and compare A-Teams to other architectures that have been used for optimization and decision support. Having introduced the A-Team architecture, we move on to describe our implementation and its key features. The implementation provides a basis for developing decision-support systems. However, creating an efficient and effective system is still more of an art than a science. To aid in the creation of A-Team systems, we present a set of principles that we have found to be useful. We conclude with an overview of A-Team based decision-support systems that have been deployed to date and discuss future research directions.

## 2. A-Team Overview

A-Teams use teams of agents to optimize a population of solutions. Agents cooperate by sharing access to populations of candidate solutions. Each agent works to create, modify or remove solutions from a population. The quality of the solutions gradually evolves over time as improved solutions are added and poor solutions are removed. Cooperation between agents emerges as one agent works on the solutions produced by another.



**Figure 1: A-Teams consist of populations of solutions and agents that create, improve and destroy solutions.**

Figure 1 presents an overview of the A-Team architecture. Each agent, shown as a block, encapsulates a particular algorithm. This algorithm may consist of a call to an external system. Within an A-Team, agents are autonomous and asynchronous. Each agent encapsulates a particular problem-solving method along with the methods to

decide when to work, what to work on and how often to work. These decisions are evaluation driven. An agent decides when to work and what to work on by looking at the evaluations of the solutions in the population. Agents are free to use any method for deciding when to work and what to work on, but intelligent strategies look at the state of the solutions in the population and take into account the agent's ability to improve them.

Agents come in three flavors: constructors, improvers and destructors. *Constructors* create initial solutions and add them to the population. *Improvers* select one or more existing solutions from the population and produce new solutions that are added to the population. Technically, improvers are modifiers and may not actually make measurable improvements in the solutions they modify. They may, in fact, make random modifications that lead to worse solutions. Such modifications may be useful in that they serve to explore the solution space and, in so doing, may lead down a path to a better solution. Typically, however, improvers encapsulate domain specific methods designed specifically to effect quick directed improvement. Finally, *destroyers* keep the size of the population of solutions in check. Their main function is to delete clearly sub-optimal or redundant solutions, while keeping promising solutions. This prevents improver agents from wasting effort by working on solutions are going nowhere.

Populations are repositories for solutions. Each population is homogenous in the sense that it holds only a single type of solution. Within a population, each solution is evaluated along a number of problem specific dimensions, resulting in an evaluation vector that determines the quality of the solution. An A-Team may contain more than one population. For example, a scheduling problem may consist of multiple sub-problems. There could be a population of solutions for each sub-problem and a population of complete solutions. The population of complete solutions can be constructed by combining solutions from the other populations. An optimization system may contain multiple A-Teams working in concert to produce complete solutions.

By sharing the set of solutions in a population, agents in an A-Team cooperate to evolve a population of good solutions. Since there is no inter-agent communication language to explicitly coordinate the behavior of sets of agents, cooperation is achieved by enabling one agent to work on the intermediate results of another. Cooperation tends to result in better solutions than could be produced by any single agent. The sequence of agent invocations that lead to the best solutions may be arbitrarily complex. (Salman *et al.*, 1997) The performance of the team, in terms of the quality and diversity of solutions and the speed with which solutions are derived, depends on the set of agents that make up the team. Later in this paper, we will have more to say about the principles that we use to create effective A-Teams.

## 2.1. Human as Agent

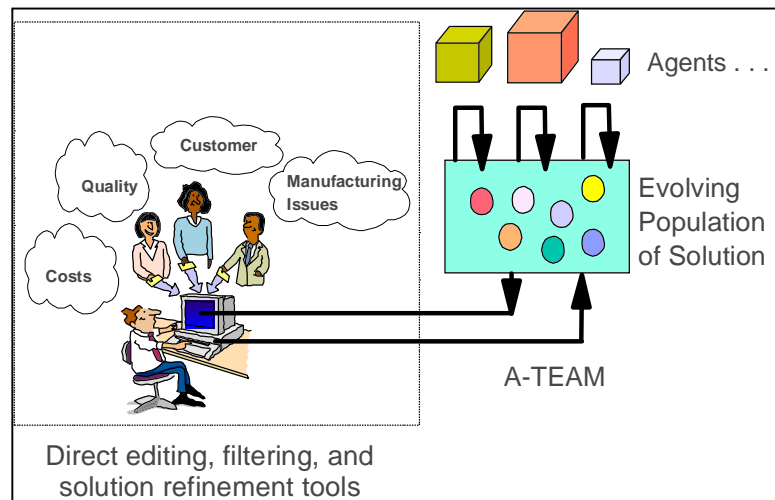
A special form of cooperation occurs between the human scheduler and the software agents (Figure 2). We have enhanced the A-Team architecture to enable humans to participate as agents so that the human scheduler can provide deep knowledge of the problem domain not captured by the software agents. Humans can also more readily adapt as circumstances of the problem change over time. By participating as agents in an A-Team, humans can generate good solutions, destroy bad ones and assist in the evaluation of solutions. Additionally, they can relax problem constraints by negotiating with concerned parties.

Complex optimization problems, such as planning and scheduling in manufacturing environments, have multiple objectives and constraints. Often, there is no single dominant algorithm that can give “the optimal” solution to the problem because of combinatorial complexity or because multiple competing objectives make it difficult to determine which solution is optimal. Heuristics for finding good solutions may be highly dependent on the specifics of the problem and may be difficult to codify. Humans can offer broad stroke guidance by seeding the population with good initial solutions. Software agents can use these solutions as a basis for further improvement. Such cooperation has been shown to be effective elsewhere (Masayuki *et al.*, 1991). Conversely, the human scheduler can refine alternatives generated by software agents so as to account for special-case considerations. The human scheduler can guide improver agents by deleting alternatives that appear hopeless. When reviewing the set of non-dominated solutions and their inherent tradeoffs, the human scheduler is best able to distinguish which alternatives are worthy of more detailed consideration. Human experts understand the implications of each constraint and can negotiate to change the problem. For example, a customer may be willing to take part of an order late as long as one truckload arrives on time. Knowing when to ask for an extension is best left to the human rather than creating a complex system that tries to capture every nuance of a problem. This approach ensures that the system will be applicable to a broader range of problems. The human scheduler thus plays a highly interactive role in our system.

Within the A-Team framework, the purpose of the utility function, which evaluates solutions, is to guide search and to identify potentially good solutions for presentation to the human decision-maker. The partial utility model must capture the high level factors that contribute to a good schedule, but it does not have to compare dissimilar measures of utility that represent tradeoffs between competing interests. We use a multi-valued utility function that evaluates a schedule along a set of dimensions. For manufacturing interests, we have found that objectives can be evaluated along four broad dimensions: Time, factors that relate to on-time delivery, Quality, factors related to product quality, Money, factors related to profitability and Disruptions, factors related to the ease of manufacture. Not coincidentally, these four measures map to the interests of the customer service representatives, quality engineers, accountants and manufacturing supervisors respectively. The utility model is used to aggregate factors that contribute to each category, but comparisons between categories

are left to the human scheduler who is presented with a set of schedules that represent the Pareto-optimal frontier. The scheduler can use these solutions as a basis for evaluating the tradeoffs, negotiating with other people in the company and coming to a final decision.

Presenting the human decision-maker with a set of alternatives that illustrate the tradeoffs to be made provides the information they need to make an informed decision. However, presenting too many solutions can overwhelm the decision-maker. In order to present a reasonably sized set of solutions, the system filters the population of solutions and presents only a subset. Currently, our systems present the set of non-dominated solutions. The decision maker can then choose any of these solutions, modify it and then put it back into the A-Team population in order to subject the alternative to further improvements by software-based agents.



**Figure 2: Decision-maker as an agent in the A-Team**

## 2.2. A-Team Advantages

The A-Team architecture offers certain key advantages:

1. *Modular:* In an A-Team, agents are autonomous and do not depend on each other for either execution or communication. Hence, in the process of software development, agents can be built independently of each other. This enables the software solution to be developed in a modular way. Agents can be easily added or deleted from the system at any time. An analysis of the population of candidate solution alternatives often indicates what type of agent should be added to improve results. The A-Team architecture enables complex systems to be implemented incrementally and more easily maintained.

2. *Distributed*: The asynchronous nature of the A-Team makes it naturally suitable for parallelism. Since agents do not depend on each other in order to function, they can be distributed over the network and run in parallel to improve performance.
3. *Robust*: System reliability is critically important in manufacturing environments. A-Teams are robust because the failure of one agent does not lead to the failure of the entire system. If under particular circumstances an agent produces an invalid solution, or fails to produce any solution, it is unlikely that all other agents will also fail.

### 3. Related Work

The A-Team architecture was originally developed by Talukdar (Talukdar et al, 1983, Talukdar et al, 1993; Talukdar et al, 1996). A-Teams have proven to be successful in addressing hard optimization problems where no dominant algorithm exists. Application areas include: nonlinear algebraic equations (Talukdar et al, 1983), traveling salesman problems (Talukdar and de Souza, 1992), configuration of task-specific robots (Murthy, 1992), design of high-rise buildings (Quadrel, 1991), control of electric networks in real-time (Talukdar and Ramesh, 1993), diagnosis of faults in power systems (Chen and Talukdar, 1993) and constraint satisfaction problems (Gorti *et al.*, 1996). More recently, our group at IBM has applied A-Teams to planning and scheduling problems in manufacturing domains including steel (Lee et al, 1996), paper (Murthy et al, 1997) and transportation scheduling (Goodwin *et al.*, 1998). We believe that the technology is generally applicable to hard optimization problems.

#### 3.1. Related Architectures

The design of the A-Team architecture was motivated by other architectures used for optimization including blackboard systems and genetic algorithms. In fact, our A-Team infrastructure could be used to implement most aspects of these other architectures. The advantage of the A-Team architecture is that it combines a population of solutions with domain specific algorithms and limited agent interaction. In addition, rich solution evaluation metrics tends to result in a more diverse set of solutions.

Blackboard architectures enable cooperation among agents, called knowledge sources, by sharing access to a common memory and allowing agents to work on different parts of the problem (Erman, Hayes-Roth, Lesser and Reddy, 1980). The key difference is that A-Teams do not have a central scheduler responsible for sequencing agent invocations. Agents are autonomous. In addition, the A-Team population typically contains many complete solutions and uses a richer set of evaluations. This tends to produce a more diverse set of solutions rather than a single solution geared towards optimizing a specific objective function.

Genetic algorithms are based on natural selection in which a population of solutions represented as bit vectors evolves by random mutation and crossover (Holland, 1975).

The solutions in an A-Team population also evolve over time, but unlike genetic algorithms, the mechanisms for creating and modifying solutions can be highly directed and take into account domain specific knowledge, rather than depending upon domain independent methods.

The ability of A-Team agents to cooperate in producing multiple candidate solutions makes it well suited for solving hard multi-objective optimization problems in a decision-support context. This is especially true in cases where the relative importance of these objectives changes constantly. (Sycara and Zeng, 1996) discuss how agent technologies can be seamlessly integrated with decision-support for information retrieval problems.

## 4. A-Team Implementation

Based on the A-Team architecture discussed previously, we have implemented an A-Team class library in order to make A-Team technology available to researchers, consultants, and third-party vendors. Our implementation provides the infrastructure needed to easily build complex and powerful optimization and decision-support applications. The implementation provides the basic components needed to create A-Teams, a configuration language for assembling and customizing the components and a user interface for interfacing with the resulting A-Team. To apply an A-Team to a particular problem the programmer need only write the problem specific code: an object to represent the problem, an object to represent a solution and procedures to encode problem specific algorithms. The library is designed using an object-oriented approach and is written in C++. It runs on multiple operating systems, including AIX<sup>1</sup>, Windows NT<sup>2</sup> and Linux. In this section, we describe the design of the class library.

At the conceptual level, our implementation consists of three components: *a graphical user interface, a class library and a configuration language* (Figure 3). In the rest of the section, we describe each of these components.

### 4.1. User Interface

An interactive, graphical user interface allows the user to view and control the execution of a set of A-Teams. The interface displays the interactions among the agents and the evolution of the solutions within the populations. The user can use the interface to control the creation, configuration, starting, and stopping of the A-Teams. The interface also allows direct interaction with a running A-Team. The user can invoke agents and remove solutions from the population. If a problem specific editor

---

<sup>1</sup> Registered trademark of IBM.

<sup>2</sup> Registered trademark of Microsoft.

is supplied, the user can also create new solutions and add them to a population or modify existing solutions.

#### 4.2. Class Library

The A-Team class library contains implementations of the basic components of an A-Team. These components provide the structure needed to create an A-Team as well as a rich set of standard behaviors. In most cases, these components provide all the non-problem specific functionality needed to create an A-Team. However, the library is organized in such a way as to allow a programmer to create specialized components that override the behavior of the standard components.

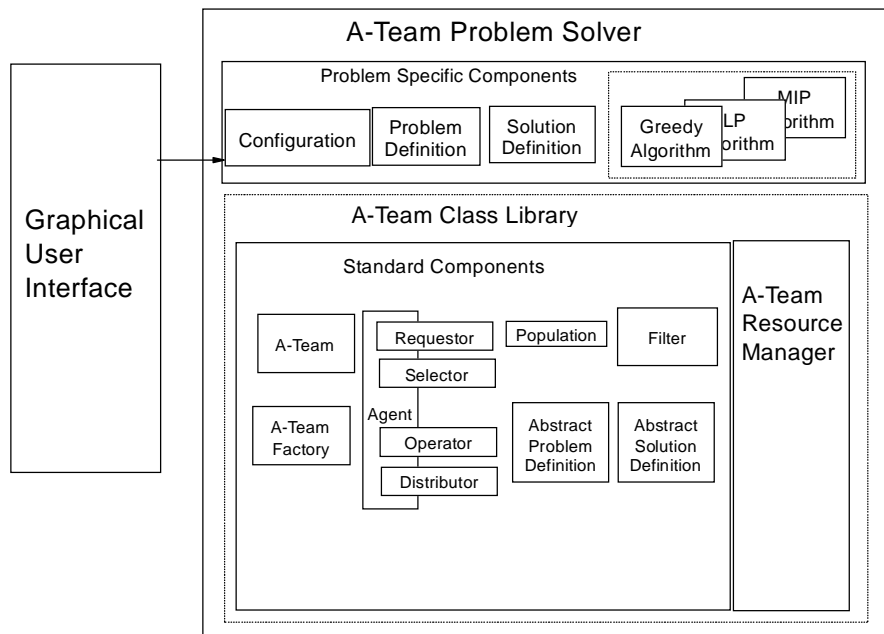


Figure 3: IBM's implementation of an A-Team framework.

The components of the A-Team library are organized in a modular fashion with well-defined roles and interfaces. Individual objects represent the agents and populations in an A-Team. An A-Team object represents an instantiation of an A-Team described by a particular configuration. This object holds the agents and populations that comprise the A-Team. New instances of an A-Team are created by a factory class. Below we describe the implementation of each of these base components and their sub-components.

*A-Team Factory:* An A-Team factory creates new instances of A-Teams. It uses an A-Team configuration, specified using the configuration language, to create instances of A-Teams. An A-Team factory creates the populations, agents and other

components that comprise an A-Team instance, it connects these components together and sets their parameters, as specified by the configuration.

- 1) *A-Team Object*: A-Team object encapsulates the components that comprise an instance of an A-Team. It provides a common interface for interacting with an A-Team and is used by the user interface and application programs. The interface provides mechanisms for starting and stopping the A-Team, for interacting with agents and for interacting with populations.
- 2) *Agents*: As well as embodying problem specific algorithms, A-Team agents must decide when to run, which solutions to work on, and what to do with any new solutions that they might generate. Correspondingly, an agent consists of four components, one component that embodies the algorithm and three decision making components, one for each decision. The decision-making components can be problem independent, while the algorithm is always problem dependent. The components of an agent are:
  - A Requestor, which requests CPU time to run the agent. A requestor is the agent's event handler. It is notified when the A-Team begins execution and can request to be notified of other events, such as a design being added to or removed from a population. Standard requestors include constructor requestors that request to run only once when an A-Team begins, improver requestors that request to run only after a population has a specified minimum number of designs and requestors with filters that request to run when the solutions in a population satisfy the filter condition. (Filters are described below.)
  - A Selector, which picks solutions from a population. Constructors use null selectors, since they do not operate on existing solutions. Improvers and destroyers use their selectors to pick solutions for modification and destruction, respectively. Standard selectors include random selection and selection from the set of solutions that pass a filter.
  - An Operator, which embodies the algorithm. For destroyers, the operator component is typically null. For constructors and improvers, the programmer must supply the operator, or this may consist of a call to an external system executable.
  - A Distributor, which decides what to do with the results of the operator. The default strategy for constructors and improvers is to add the resulting design to the output population. The distributor for a destroyer deletes the resulting solution from the population.
- 3) *Populations*: A Population is a repository for solutions. It provides mechanisms for adding, deleting and copying solutions. It can also return a set of solutions that pass a given filter.
- 4) *Filters*: A filter embodies a test of a solution and when applied to a solution, returns either true or false. As specified above, filters are used to select solutions

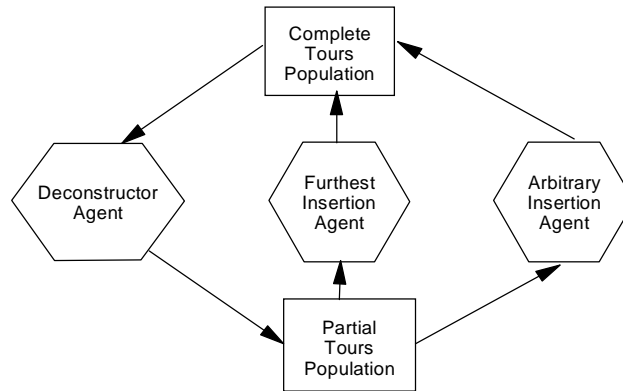
from a population and to gate the behavior of other components. Our implementation includes a large set of domain independent filters. We have filters that find the non-dominated set of solution, the NDFilter, filters that return the set of solutions with evaluations that exceed a given threshold, the ThresholdFilter and filters that find duplicate solutions, based on their evaluations, the DuplicateFilter. In addition, a set of composite filters allows the programmer to combine other filters in useful ways. The composite filters include the AndFilter, the OrFilter, the NotFilter, and the XorFilter. We can combine the standard filters to create new filters. For example, we could use the NDFilter, the DuplicateFilter and the AndFilter to construct a filter that returned duplicate, non-dominated solutions. In addition to the standard filters, programmers can create problem specific filters that test specific aspects of a solution.

- 5) *A-Team Resource Manager*: Even though A-Team agents run asynchronously, some mechanism is needed to assign resources to agents that request to run. In an implementation that uses a different thread for each agent, the operating system scheduler could perform this function. In our implementation, we use a resource manager to implement resource allocation policies and to create an abstract interface to the underlying hardware and operating system. This allows us to run an A-Team on a uni-processor, a parallel processor or on a distributed architecture, like CORBA. Our implementation currently includes a uni-processor resource manager and a CORBA resource manager. The uni-processor manager uses a single thread of control and randomly selects an agent to run from the list of agents waiting to run. Problem specific resource allocation strategies and resource managers that learn to allocate more resources to agents that perform well have also been implemented, but are not part of the standard library yet.
- 6) *Abstract Definitions of Problem Specific Components*: Our implementation includes abstract placeholders for problem specific components. These abstract components can be specialized to create the components needed to represent the problem definition object and the solution definition object.

### **4.3. A-Team Configuration Language**

The A-Team configuration language allows the user to specify the components, interconnections and parameters for an A-Team. The A-Team factory uses this specification when constructing an A-Team. Use of the configuration language reduces the amount of code that must be written and allows A-Teams to be rapidly reconfigured without recompiling. The user interface and the configuration language enable the user to dynamically re-configure A-Teams without the need to write C++ code.

The language is relatively simple, but extensible. It can be used specify all the standard components of the A-Team library and set their parameters. It can also be used to specify user-defined objects and set their parameters. Below, we give a small excerpt from a sample configuration file to give a flavor of the language.



**Figure 3: The connectivity between three of the agents and two of the populations in an A-Team to solve the Traveling Salesman Problem.**

```

1   A tspAteam ATEAM ISA ateam
2     WITH
3     agents = (tspDeconstructorAgent, tspFurthestInsertionAgent,
4               tspArbitraryInsertionAgent, tspDestroyerAgent);
5     populations = (tspCompleteToursPop, tspPartialToursPop);
6
7     resource = (tspResource);
8
9     connect = (tspDeconstructorAgent, input, tspCompleteToursPop) ;
10    connect = (tspDeconstructorAgent, output, tspPartialToursPop);
11
12    connect = (tspFurthestInsertionAgent, input, tspPartialToursPop);
13    connect = (tspFurthestInsertionAgent, output, tspCompleteToursPop);
14
15    connect = (tspArbitraryInsertionAgent, input, tspPartialToursPop);
16    connect = (tspArbitraryInsertionAgent, output, tspCompleteToursPop);
17
18    connect = (tspDestroyerAgent, input, tspCompleteToursPop);
19  END
20
21  A tspResource RESOURCE ISA UniResourceAllocator
22    WITH
23    maxInvocations = (1000);
24    trace = (TRUE);
25  END
  
```

This excerpt is a segment from the configuration file of a Traveling Salesman Problem (TSP) solving A-Team. Line 1 specifies that the component being described is an ATEAM that is implemented using the standard ateam object. Between the “WITH” and the “END” come the parameter settings, in this case a list of agents, populations, the connections between agents and populations, and a resource manager. In this

example, there are four agents and two populations. One population contains complete solutions to the TSP problem and the other contains tour fragments. The deconstructor agent takes complete solutions and extracts efficient sub-tours and puts them into the population of tour fragments. The other two agents take zero, one or two tour fragments and create complete solutions, which are added to the `tspCompleteToursPop`. The definition of the `tspResource` begins on line 26. Here the resource manager is defined as a `UniResourceAllocator`. The maximum number of agent invocations is set to 1000 and tracing of agent invocations is turned on. Further details of agents and populations are defined later in the configuration file.

## 5. Building Effective A-Teams

The best A-teams are those that produce diverse optimal or near-optimal solutions quickly and consistently. The design of an A-Team is therefore a multi-objective optimization problem. Although diversity is only important in the context of multiple objectives, the cooperative behavior of A-Team agents is still valuable even if there is a single objective, or fitness function. (Huberman, Lukose, & Hogg 1997) offer insight into why combining multiple algorithms produces solutions that are preferable to the solutions generated by individual algorithms operating alone, and offers a measure for the degree of cooperation involved based on statistical correlation of performance. This is analogous to the way in which machine learning researchers employ ensembles of classifiers to produce higher levels of accuracy than can be achieved using any single classifier (Dietterich, 1997).

Although there is no formal theory that describes how to create an effective A-Team, our experience developing real-world applications has lead us to discover a set of guiding principles. Much of our on-going research focuses on A-Team performance measures that we use to discover good agent portfolios.

- Too great a dependence on computationally intensive agents results in fewer solutions and less agent cooperation. The result is an A-Team that takes longer to run while producing less varied results.
- Relying too heavily on simple solution-tweaking agents can result in clusters of solutions each consisting of many nearly identical solutions. In addition, simple agents may not make sufficient progress towards optimality.
- Destroyers provide selective pressure by encouraging agents to work on promising solutions and therefore critical. However, aggressive destruction can also decimate population diversity, undermining the variability of candidate alternatives available for decision-support purposes.

## 6. IBM Products That Use A-Teams

Using our A-Team architecture, we have built applications for paper mill scheduling, steel mill scheduling, transportation scheduling and wafer start planning for semiconductor manufacturing. These solutions have been successfully fielded at more than a dozen sites and are currently being sold worldwide. In this section, we describe some of these products.

IBM Trim<sup>3</sup> Optimization system (product no. 5799-A11) is a software optimization tool that gives paper companies the ability to maximize their manufacturing yield while satisfying customer requirements. The software provides multiple trim alternatives, with each alternative trim solution evaluated according to several evaluation criteria specified by the user. The optimizer combines a powerful suite of mathematical approaches and heuristics within the A-Team framework to provide to the human schedulers a diverse set of near-optimal trimming alternatives.

IBM Load Planning and Distribution optimization software application (product no. 5799-A12) provides state-of-the-art tools for distribution planners. It is designed to minimize cost and maximize vehicle utilization while maintaining on-time delivery of products. The software considers production schedules, delivery commitments and shipment methods to produce optimized distribution schedules. We address this multi-criteria optimization problem by combining various mathematical approaches such as linear programming, integer programming and domain specific heuristics (Goodwin et al 98).

IBM Global Scheduling Optimization software (product no. 5799-A13) provides integrated enterprise-wide scheduling solutions for mills. The software produces complete production and distribution schedules for the entire enterprise which includes allocation of orders to mills and machines, grouping and sequencing orders into batches for production, trimming the for optimal manufacturing yield and producing optimal transportation schedules for the produced items. We have used the A-Team architecture in multiple layers in our global optimization software and it embodies a rich set of problem solving methods from multiple areas such as operations research and artificial intelligence.

Companies have reported substantial benefits using these technologies. One paper customer has reported a 2 inch increase in the utilization of their reels using IBM Trim Optimization along with a 10% reduction in transportation costs using IBM Load Planning technology, recouping their investment in less than one year. These financial benefits are in addition to improvements in manufacturing yield and customer service (Shaw, 1998; Hoffman, 1996).

---

<sup>3</sup> The word "Trimming" in paper manufacturing terminology refers to cutting paper machine reels into rolls of smaller diameter and width so as to satisfy customer demand while maximizing utilization of the reel. See (Murthy et al. 97) for further details.

## 7. Conclusions and Future Work

We have described an agent-based architecture known as an A-Team that is now proving itself useful for addressing real-world optimization problems. An A-Team consists of a set of configurable agents and a set of populations that serve as repositories for candidate solutions. A-Teams are simple yet effective. They are simple because there is no inter-agent communication. Agents cooperate by having shared access to the solutions in a population. Nevertheless, this simple form of cooperation makes the A-Team architecture very powerful. Our A-Team implementation makes it particularly easy to incorporate unique problem-solving methods that could be custom written for the problem or even previously developed object code. A-Teams are useful for multi-objective optimization and decision-support because a set of agents can embody many different objectives and are more readily able to produce a diverse set of solution alternatives. We have discussed how our implementation further extends prior work with A-Teams by enabling the human decision-maker to act as an additional agent, able to interact with other agents by creating, refining, and destroying solution alternatives. We have also explained why A-Team-based systems are modular, robust and inherently easy to parallelize.

Building effective A-Teams depends upon the mix of agents and their individual capabilities. As part of our ongoing research, we are continuing to investigate ways of creating good combinations of agents using quantitative measures of A-Team performance. We are also investigating whether agents capable of learning and other adaptive behaviors can further improve system performance.

We are continuing to enhance our A-Team architecture, and are making the A-Team class library available to researchers, consultants and third party vendors. For information on obtaining the latest release of the A-Team library, please contact the authors.

## References

- Chen, C.L., Talukdar, S.N.; 1993. *Causal Nets for Fault Diagnosis*. 4th International Conference on Expert Systems Application to Power Systems, Melbourne, Australia, Jan 4-8.
- Dietterich, T.; 1997. *Machine-Learning Research: Four Current Directions*. AI Magazine **18**(4): 97-136.
- Erman L. D., Hayes-Roth F. Lesser V. R. and Reddy R. D; 1980. *The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty*. In ACM Computing Surveys 12(2).
- Finin T., Labrou Y., and Mayfield J.; 1997. *KQML as an Agent Communication Language*. In Software Agents. Meno Park, AAAI Press.
- Goodwin R. T., Rachlin J., Murthy S., Akkiraju R.; 1998. *Interactive Decision Support: Advantages of an Incomplete Utility Model*. AAAI Spring Symposium on Interactive and Mixed Initiative Decision-Theoretic Systems.

- Gorti, S. R., Humair S., Sriram, R.D., Talukdar S., and Murthy S.; 1996. *Solving Constraint Satisfaction Problems Using A-Teams*. Artificial Intelligence for Engineering Design, **10**:1-19, Cambridge University Press.
- Hoffman T; 1996. *A.I. Based Software Models Help Cut Production Costs*. Computer World September 2, 1996. [http://www.computerworld.com/idx\\_usea.htm](http://www.computerworld.com/idx_usea.htm).
- Holland J. H; 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor MI.
- Huberman, B., Lukose, R., and Hogg, T.; 1997. *An Economics Approach to Hard Computational Problems*. Science **275**:51-54.
- Lee, H.S., Murthy, S., Haider, S.W., and Morse, D.; 1996. *Primary Production Scheduling at Steel-Making Industries*. IBM Journal of Research and Development.
- Masayuki, N. and Morishita, S.; 1991. *Cooperative scheduling and its application to steelmaking processes*. IEEE Trans. on Industrial Electronics. **38**(2).
- Murthy, S. Synergy in Cooperating Agents; 1992. *Designing Manipulators from Task Specifications*. Ph.D. Dissertation, Carnegie Mellon University.
- Murthy, S., Rachlin, J., Akkiraju R., Wu F.; 1997. *Agent-Based Cooperative Scheduling*. In Constraints & Agents. Technical Report WS 1997-97-05. Menlo Park:AAAI Press.
- Quadrel R.; 1991. *Asynchronous Design Environment: Architecture and Behavior*. Ph.D. Dissertation. Carnegie Mellon University.
- Salman F., Kalagnanam J., Murthy S.; 1997. *Cooperative Strategies for Solving the Bicriteria Sparse Multiple Knapsack Problem*. IBM Research Report RC 21059(94164).
- Shaw M.; 1998. *Madison Streamlines Business Processes with Integrated Information System*. Pulp and Paper, Volume 72, Issue 5.
- Sycara K, and Zeng D.; 1996. *Coordination of Multiple Intelligent Software Agents*. International Journal of Cooperative Information Systems. **5**(2 & 3)
- Talukdar, S.N., Baerentzen, L., Gove, A., and Souza, P.; 1996. *Cooperation Schemes for Autonomous Agents*. Times Assincronos para Problemas Industriais, Sao Paulo.
- Talukdar S.N., Pyo S.S., and Mehrotra R.; 1983. *Distributed Processors for Numerically Intense Problems*. Final Report for EPRI Project. **RP** 1983-1764-3
- Talukdar S.N., and Ramesh V.C.; 1993. *Cooperative Methods for Security Planning*. 4th International Conference on Expert Systems Application to Power Systems. Melbourne, Australia, Jan 4-8.
- Talukdar S.N., and Souza P.S. de.; 1992. *Scale Efficient Organizations*. In Proceedings of the 1992 IEEE International Conference on Systems, Man, and Cybernetics. Chicago, Illinois, Oct. 18-21.
- Talukdar, S.N., Souza, P. de, and Murthy S.; 1993. *Organizations for Computer-Based Agents*. Engineering Intelligent Systems, 1(2).