Proceedings of the Artificial Neural Networks in Engineering (ANNIE Conference, St. Louis, MO, Vol. 7, pp. 73-78, November, 1997

ASYNCHRONOUS TEAMS (A-TEAMS) AND THE A-TEAMS TOOLKIT: AN AGENT-BASED PROBLEM-SOLVING ARCHITECTURE AND SOFTWARE FRAMEWORK

PHILIP CHANG, JOHN DOLAN, JAMES HEMMERLE, MICHAEL TERK, AND SAROSH TALUKDAR

The Engineering Design Research Center and The Robotics Institute

Carnegie Mellon University, Pittsburgh, PA

ABSTRACT:

This paper presents a biologically inspired architecture for problem solving called Asynchronous Teams (A-Teams) and a Toolkit for rapid assembly and prototyping of A-Teams. A-Teams are distributed, cooperative, and scale-efficient agent-networks. We define an "agent" as anything that can act, sense, and exert some control over its actions. A-Team agents are completely autonomous, that is, each agent has exclusive control over its actions. The strengths of A-Teams in problem solving arise from agent cooperation, agent distribution, and scale efficiency. Agent cooperation produces better results than can be achieved by individual agents, often leading to optimal results. A distributed architecture provides autonomy without resource constraints and control dependencies, and makes new agents relatively easy to add. Finally, scale efficiency means that the more agents that are added, the better the results in terms of solution quality and speed. The A-Teams Toolkit greatly facilitates the formation of A-Teams and provides a general software framework for distributed problem solving.

INTRODUCTION

An A-Team (Talukdar, Baerentzen, Gove, and de Souza, 1995, 1996) is a scale-efficient network of distributed computer agents working together to solve a difficult problem. A-Teams, which are biologically inspired, are characterized by autonomous agents and cyclic data flow. A-Teams have several unique strengths for problem solving: agent cooperation, whereby agents complement one another to produce better solutions than any one agent could achieve; a distributed character, which allows graceful degradation when agents cease to be useful or to function; and scale-efficiency, which allows the size of an A-Team to be adjusted to the size of the problem.

Talukdar and de Souza (de Souza and Talukdar, 1991) introduced the term Asynchronous Teams to represent a team of asynchronous algorithms, or agents. They used Newton-Raphson and Genetic Algorithms (GA) as agents in an A-Team to solve nonlinear algebraic equations in a shorter time than when the methods ran individually. Quadrel (Quadrel, 1991) used A-Teams to handle multiple objectives and constraints in high-rise building design. Tsen (Tsen, 1995) used A-Teams to solve train scheduling problems involving two different types of solution representation. Table 1 summarizes problems to which the A-Teams methodology has been successfully applied.

TABLE 1. PAST A-TEAMS APPLICATIONS (IN ORDER OF TIME OF COMPLETION)

Domain	Lesson Learned
Non-linear algebraic equations	Proof of A-Team concept
Traveling Salesman Problem	Scale-efficiency
Hi-rise building design	Handling multiple, conflicting criteria
Robots-on-demand	Balancing construction with destruction
Power system diagnosis	Quantitative Bayesian networks
Power system control	Global optimization strategies, formations
Train scheduling	Multiple representations

MOTIVATION

Although many powerful algorithms and heuristics have been developed and implemented in software in the A-Teams work described above, and in the history of optimization and problem solving, these implementations are rarely reused due to their *ad hoc* nature. We have created the A-Team Toolkit (Chang, Dolan, and Terk, 1996) in order to reduce the existing barriers for reuse and provide auxiliary tools for rapid prototyping and data collection. The Toolkit was designed to meet three goals: 1) a distributed agent architecture, enabling parallel processing and thus greater agent autonomy and scale-efficiency; 2) modularity, so that algorithms, data structures, and communications are cleanly separated from one another and existing algorithms can be easily incorporated; and 3) real-time adaptability and monitoring through a convenient, point-and-click, icon-based graphical user interface (GUI).

DESIGN AND IMPLEMENTATION

In the Toolkit, the high-level objects that are configured to form an A-Team are memories and agents. Each memory contains objects called solutions, and each agent is composed of four objects: a scheduler, searcher, selector, and operator (see Figure 1). Using this terminology, an A-Team operates by allowing agents to check solutions in and out of memories and to modify these solutions.

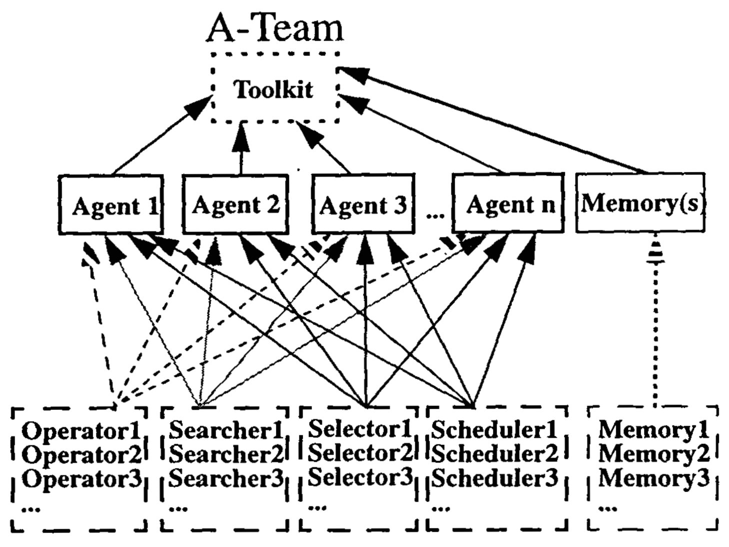

Figure 1: Libraries of agent and memory components.

Each solution contains four pieces of information: *ID*, *data*, *evaluation*, and *history*. A unique solution *ID* allows tracking of the solution. The *data* portion is a user-defined, application-dependent data structure which expresses the solution. In the Traveling Salesman Problem (TSP), for example, this structure might be a vector of integers giving the city numbers in the tour. The *evaluation* is an array of floating point numbers, each of which corresponds to the result of a different evaluation function. In the TSP, a straightforward evaluation is the tour length. The *history* contains useful statistical information about the solution, such as which agent created it, how many times it has been read, etc.

Memories are objects which store solutions, manage requests for solution check-in and check-out, and keep track of individual and collective solution histories. Agents are objects which operate on solutions using the functionality provided by the four components of which they are composed. The scheduler determines when an agent should run. The searcher sends a request for solutions meeting certain criteria to a memory; upon receiving the solution IDs, the agent's selector picks one solution and requests it from the memory. In general, the agent's central part, the operator, then acts on this solution in a certain way.

The Toolkit currently supports six agent types: seeder, destroyer, modifier, evaluator, transfer, and monitor. Four of these agent types correspond to the operator type; however, destroyers and transfer agents have no operators. A seeder creates new solutions, whereas a destroyer removes them. A modifier requests a solution from a memory, makes changes to its *data* portion, and reinserts it with a new solution ID. An evaluator applies an arbitrary evaluation function to the *data* portion of a solution and returns a single float type value to the memory to be inserted into the solution's evaluation array. A transfer agent moves a solution from one memory to another without modification. A monitor provides information on the performance of the A-Team. There are currently three defined monitor types: memory monitors, displayers, and recorders. Memory monitors display memory statistics, including solution history, the number of solutions in memory, and the number of evaluated solutions. Displayers graphically depict A-Team performance. Recorders store performance statistics to file.

The described methodology was implemented in the C++ language. Several standard tools were additionally used in order to facilitate development. Tk/Tcl, a C-compatible scripting language, was used to create a point-and-click, icon-based graphical user interface. PVM (Parallel Virtual Machine) was used for communications among agents, memories, and user interface on multiple machines. LEDA (Library of Efficient Data types and Algorithms) was used for solution management and searching.

EXPERIMENTS

Experiments were conducted on the performance, efficiency, and flexibility of the Toolkit. The goal was to determine the extent to which use of the Toolkit 1) improves solution quality, 2) speeds the process of assembling A-Teams, and 3) enables the encapsulation of legacy code. Tests were conducted on two problems: the ATT532 TSP (Padberg and Rinaldi, 1987), which seeks the shortest tour of 532 American cities, and a train scheduling application on the Burlington Northern Sand Hills line which seeks to minimize tardiness. The ATT532 problem has a known optimal tour length of 27686 units. The minimum tardiness for the Sand Hills line achieved by the system currently used by the railroad, the Deadlock Prevention Mechanism (DPS), is 212 minutes. In seeking to improve solution quality, the Toolkit's rapid prototyping capability was used on both problems to vary agent types, data flow, memory size, initial seeding size, and the number of machines. The memory population sizes for the experiments were in the range of 25 to 1600 solutions. The number of initial seed solutions varied from 20 to 1000. Each experiment ran from 30 to 60 minutes. To test the efficiency of the Toolkit, a comparison was made between the time needed to assemble an A-Team for the TSP with and without the Toolkit. To test the flexibility of the Toolkit, a single-process A-Team implementation of a train scheduler by Tsen (Tsen, 1995) was encapsulated within the Toolkit framework.

TSP Application

Various combinations of seeders, modifiers, evaluators, and destroyers were used to solve the TSP. Three seeders were used: random, arbitrary insertion, and branch and bound. Five modifiers were used: Lin-Kernighan (the most powerful single TSP algorithm), two-opt, intersection, arbitrary insertion, and kick (see de Souza, 1993 for more details). One evaluator was used to calculate the length of a tour for a given solution. A destroyer was used which deleted the solution with the longest tour length. Several monitor agents displayed the A-Team's performance. Performance testing on the TSP produced the following results: 1) the wider the variety of agents, the better the performance, a confirmation of the principles of scale efficiency and specialization; and 2) distributing agents and memories on *n* machines increased solution speed by more than *n*-fold in the cases tested. This superlinear increase is due to the increased ability of agents to benefit from one another's solutions. The speed increase achieved by a given number of multiple machines could be further improved by 7% on average through the addition of load balancing.

The Toolkit reduced the time needed to assemble an A-Team for the TSP by an order of magnitude. The original implementation of an A-Team for the TSP (de Souza, 1993) took approximately two thousand hours over a period of 18 months. Most of that time was spent was on developing communications data structures and protocols, and building auxiliary tools which are already present in the Toolkit infrastructure. The Toolkit version of the TSP required about two hundred hours, and primarily involved the software encapsulation.of agents and memories in Toolkit-compatible formats.

Train scheduler application

A single-process version of the train-scheduling line planner written by Tsen (Tsen, 1995) was broken up and used to test the capability of the Toolkit to encapsulate legacy code. Once complete, the Toolkit version of the line planner consisted of one memory (storing priority matrices and lateness evaluations), sixteen modification operators, twelve identical evaluation operators, a random creator (to initially seed the memory), one destroyer, and a string chart generator.

The Toolkit version had lateness results in the same range as the single-process runs, demonstrating the ability to encapsulate legacy code without losing functionality. In addition, some Toolkit results were better than the best achieved by the single-process version. Tsen (Tsen, 1995) reported the best lateness after running the single-process line planner application for 15 minutes was 83 minutes, whereas the best lateness obtained by the Toolkit on the same scenario was 61.7 minutes. We attribute this improvement to the ability afforded the experimenter by the Toolkit to explore many more configurations in a given period of time than is possible in a manually constructed version of the line planner, in which the overhead between runs is much greater. This result shows that even with algorithms and heuristics that have been in use for many years, there is still potential for improvement in solution quality and speed by allowing rapid reconfiguration to discover effective coordination strategies.

CONTRIBUTIONS AND FURTHER WORK

The future of distributed problem solving lies in the ability to capitalize on the vast number of algorithms and heuristics already written by researchers all over the world. The A-Teams concept addresses this problem by providing a general agent-based problem-solving methodology capable of simultaneously harnessing the power of numerous, disparate problem-solving agents. Beyond this, the A-Teams Toolkit provides a software framework for the encapsulation, rapid prototyping, and testing of these distributed aggregates of problem-solving agents with the following features: 1) standards for encapsulation of algorithms and data structures; 2) an infrastructure for the rapid prototyping of distributed problem-solving, including libraries of agents and memories, transparent communication protocols, and load balancing among machines; and 3) a GUI enabling the dynamic instantiation and reconfiguration of agents and memories and monitoring for dynamic display and update.

There are several areas of further work to improve the Toolkit's capabilities and usefulness. Encapsulation of algorithms for Toolkit-compatibility and assembly of agents from pre-compiled scheduler, searcher, selector, and operator components, which is currently done manually, should be automated to the greatest degree possible. Additional schedulers and searchers supporting more complicated search descriptions should be developed. Allowing the human user to interact with the Toolkit as another agent, capable of altering, rather than simply monitoring, solutions in real-time, would increase the power of the system. Finally, to make A-Teams widely available to the research community, we would like to provide World Wide Web-available database registries giving locations and capabilities of the various components (algorithms, heuristics, memory types, schedulers, etc.) already encapsulated for the Toolkit.

REFERENCES

Philip Chang, John Dolan, Michael Terk, Asynchronous Team Toolkit User's Guide, Carnegie Mellon University, April 1996.

Pedro S. de Souza, Sarosh N. Talukdar, Genetic Algorithms in Asynchronous Teams, Proceedings of the Fourth International Conference on Genetic Algorithms, Los Altos, CA, 1991.

Pedro Sergio de Souza, "Asynchronous Organization for Multi-Algorithm Problems", Ph.D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, April 1993.

Seshashayee S. Murthy, Synergy in Cooperating Agents: Designing Manipulators from Task Specification, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, February 1992.

M. Padberg, G. Rinaldi, Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut, Operations Research Letters, Volume 6, Number 1, March 1987.

S. S. Pyo, Asynchronous algorithms for Distributed Processing, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, February 1985.

Richard W. Quadrel, Asynchronous Design Environments: Architecture and Behavior, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1991.

V.C. Ramesh, "Initial Search and Asynchronous Decomposition", Ph.D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, June 1994.

Sarosh Talukdar, Asynchronous Teams and Machine Cooperation, 1995.

Sarosh Talukdar, Lars Baerentzen, Andrew Gove, Pedro de Souza, Cooperation Schemes For Autonomous Agents, Engineering Design Research Center, Carnegie Mellon University, 1996.

Chung Kang Tsen, "Solving Train Scheduling Problems using A-Teams", Ph.D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, June 1995.