Improving Cooperation Among Autonomous Agents in Asynchronous Teams

Lars Baerentzen, Sarosh N. Talukdar

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh PA 15213

12 March 1997

Abstract

An asynchronous team (A-Team) consists of autonomous agents that cooperate by working on one another's results. For the cooperation to be effective, the agents must pick the right results to work on. This is the selection problem. We develop a theoretically well-founded method for addressing this problem. It relies only on data from prior applications of the A-Team, and on an oracle for answering yes/no to the question: "Is this solution an acceptable final solution?" The method is useful when selection cannot be based on observable attributes of the solutions, either because the solutions are prohibitively expensive to evaluate or because there is no correlation between their attributes and the usefulness of the agents. The method is illustrated with an A-Team for hierarchical scheduling in a jobshop with exotic machines.

1. Introduction

An A-Team is a problem-solving organization consisting of memories and autonomous agents connected into a strongly cyclic network. Each memory is dedicated to a subproblem of the overall problem. Trial solutions to this subproblem collect in the memory to form a population. Each agent operates between two memories, reading solutions from the first, modifying them and writing results to the second memory. Agents cooperate by modifying one another's modifications. Being autonomous, they choose which results to modify and when.

The agents in an A-Team can be divided into two types: Construction agents that produce new solutions, and destroyers that erase solutions. An agent consists of several parts.

- an operator, that, in a construction agent, transforms objects from its input-memory into objects that are placed in its output-memory; and in a destruction agent erases objects.
- a scheduler, that decides when the agent should run.
- a selector, that decides which object from the input-memory the operator should work on.

A more detailed description of A-Teams and a protocol for designing them can be found in [1].

1.1. Selection

From the agents' point of view the selection problem is: "Which solution should I read in and transform?" From the solution's point of view, the selection problem is: "Which agent should modify me?" And after that: "Which agent should work on my modification?" More generally, the selection problem is: "What are good sequences of agents to apply, when each agent in the sequence works on the result of the previous agent?" If there were a way of determining which agent to apply from looking at the observable attributes of a solution, the selection problem would be straightforward. We are interested in a systematic method for determining such sequences when virtually no information about a particular solution is observable, but the behavior of the entire A-Team has been observed through a large number of previous runs for a number of problem instances.

We assume the following:

1. There is an oracle which given a solution answers yes/no to the question: "Is this solution an acceptable final solution?"

2. No other observations can be made about a solution.

3. We have a number of historical sequences of agents that transformed an initial solution into an acceptable final solution.

From a statistical point of view we can expect short historical sequences, and subsequences of short sequences to be better on average than long historical sequences and subsequences of long sequences. We want to determine the probability that a given sequence of agents when applied to an initial solution will result in an acceptable final solution.

We want a compact way of storing the information about good sequences as well as information about which sequence of agents has been applied to each solution in the memory so far. We want to store this information within each solution in the memory. This allows the agents to maintain autonomy in deciding which solution to select. They can use the provided information in their selection or they can completely ignore it should they so desire. In most practical cases some of the agents may be able to observe more information about a particular solution. In such case it is highly desirable to avoid having a central controller decide which sequence of agents to apply.

To find good sequences of agents, we develop a parametric model for how a solution is modified by agents reading and writing to the same memory. We then use maximum-likelihood-estimation to fit the parameters of our model to the observed successful sequences.

2. The Model

Let:

S be the space of all solutions that can be stored in our memory.

G (Goal) be the subset of S that we consider acceptable final solutions.

S_0, S_1, S_2,... be subsets of S such that:

- $S_0 = G$

- S_1 contains those non-acceptable solutions for which there exists some agent that will turn the solution into an acceptable final solution with positive probability.

- S_{n+1} contains those solutions not in any earlier S_k ($k \le n$) for which there exists some agent that will turn the solution into a solution in S_n with positive probability.

A solution in S_n be said to be at a distance n from the Goal.

Let us make the following assumptions:

1. If $x \in S_n$ and agent A modifies x then the result A(x) is at most one step further away from the goal, that is $A(x) \in S_{n-1} \cup S_n \cup S_{n+1}$. This automatically holds if all agents have an inverse.

2. The part of the solution space S that is not contained in G or in some S_n is made inaccessible by destroyers or is empty to begin with. That is, from any solution allowed into the memory there is a finite sequence of agents that has positive probability of transforming that solution to a solution in G.

Definitions:

- A solution is called an initial solution if it is not the result of any agent modifying some other solution.

- The pre-sequence of a solution is the sequence of agents that generated it from some initial solution. Initial solutions have the empty pre-sequence.

- A pre-sequence of $A_{\sigma(1)}, A_{\sigma(2)},..., A_{\sigma(n)}$ for a solution x means that x was generated from an initial solution y by first applying agent $A_{\sigma(1)}$ then applying $A_{\sigma(2)}$ to $A_{\sigma(1)}(y)$, etc.

When we can only observe whether a solution is an acceptable final solution or not, the only other thing we can condition on when trying to estimate the usefulness of applying a particular agent to a particular non-acceptable solution is the pre-sequence of the solution. Ideally we would like to know:

- The probability of a solution being in S_n conditioned on its pre-sequence.

- The probabilities for each agent A that A(x) will be closer to G, further away from G or at the same distance to G conditioned on the pre-sequence of x.

To keep the model tractable however, we must settle for the following approximation. We specify the above probabilities only for initial solutions, and then we specify matrices for calculating the above probabilities for a solutions with pre-sequence $A_{\sigma(1)},..., A_{\sigma(n-1)}, A_{\sigma(n)}$ from the probabilities for a solution with pre-sequence $A_{\sigma(1)},..., A_{\sigma(n-1)}$.

- Let π_n be the probability that an initial solution is in S_n (n = 0, 1, 2,..., N)

- Let p_k, q_k and r_k ($p_k + q_k + r_k = 1$) be the probabilities that govern how agent A_k modifies an initial solution x. That is, let:

  p_k = Pr (A_k(x) is closer to G than x)

  q_k = Pr (A_k(x) is further away from G than x)

  r_k = Pr(A_k(x) is at the same distance from G as x)

- Let:

$$M^{i,j} = \begin{bmatrix} m_{pp} & m_{pq} & m_{pr} \\ m_{qp} & m_{qq} & m_{qr} \\ m_{rp} & m_{rq} & m_{rr} \end{bmatrix} \tag{1}$$

  be a matrix describing how running agent A_i modifies the probabilities for agent A_j. In particular: If a solution x with some pre-sequence $A_{s(1)},..., A_{s(n-1)}$ has probabilities (p, q, r) of being brought closer to, of being moved further away from, and of staying at the same distance from the goal, respectively, when agent A_j is applied then after A_i has been applied $A_i(x)$ will have probabilities:

$$(p, q, r)^{new} = (p, q, r)M^{i, j} \tag{2}$$

for being brought closer to, for being brought further away from and for staying at the same distance to the goal when A_j is applied.

The set of parameters π_n (n = 0,1, 2,...), p_k, q_k and r_k (for each agent A_k) and $M^{i,j}$ (for each pair of agents A_i and A_j) completely specifies our model. Given values for all the parameters it is straight-forward to compute the probability of a solution being in S_n conditioned on its pre-sequence, as well as the probability of agent A_k bringing the solution closer to or further away from the goal.

There are however cases where the model is not adequate. If the A-Team is such that applying agent A_1 to an initial solution x has probability 1 of bringing it closer to the goal and the same is the case for the new solution $A_1(x)$ then by (2) the first row of $M^{1,1}$ must equal [1,0,0]. Hence as long as we continue to only apply agent A_1 the solution will always be brought closer to the goal. So there is no set of parameters for our model that allows us to model a situation where the first 5 times agent A_1 is applied the solution is always brought closer to the goal, but after that agent A_1 will only bring the solution closer to the goal with probability say 1/2.

3. Estimating the Model

Given specific values for our parameters:

$$\Theta = \{\pi_n, n=0,...,N; (p_k, q_k, r_k), k=1,...,\# \text{ agents}; M^{i,j}, i=1,..., \# \text{ agents}, j=1,...,\# \text{ agents}\}$$

and given a sequence of agents $A_{\sigma(1)}, A_{\sigma(2)},..., A_{\sigma(T)}$, we can calculate the probability that the observed sequence leads to an acceptable solution and that no earlier solution in the sequence was acceptable using the following algorithm.

1. initialize: $prob_0[n] := \pi_n$; $(p_0[k],q_0[k],r_0[k]) := (p_k, q_k, r_k)$
2. for t := 1 to T do:
3. $prob_{t-1}[-1] := 0$; $prob_{t-1}[0] := 0$; $prob_{t-1}[N+1] := prob_{t-1}[N]$
4. for each n=0,...,N:

$$prob_t[n] = prob_{t-1}[n-1] * p_{t-1}[\sigma(t)] +$$
$$prob_{t-1}[n] * r_{t-1}[\sigma(t)] +$$
$$prob_{t-1}[n+1] * q_{t-1}[\sigma(t)]$$

5. for each agent k: $(p_t[k],q_t[k],r_t[k]) := (p_{t-1}[k],q_{t-1}[k],r_{t-1}[k]) * M^{\sigma(t),k}$
6. next t
7. return $prob_T[0]$ as the probability that we reach an acceptable solution for the first time after applying the entire sequence $A_{\sigma(1)}, A_{\sigma(2)},..., A_{\sigma(T)}$ to an initial solution.

Given we have an observation consisting of several sequences that led to an acceptable solution:

$$\Sigma = \{\{A_{\sigma_1(1)}, A_{\sigma_1(2)}, \ldots, A_{\sigma_1(T_1)}\}, \ldots, \{A_{\sigma_m(1)}, A_{\sigma_m(2)}, \ldots, A_{\sigma_m(T_m)}\}\}$$

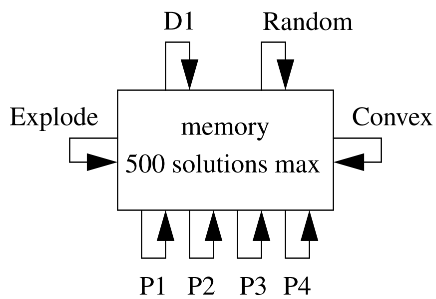
we can calculate the likelihood of getting this observation given our parameters:

$$L(\Theta, \Sigma) = \prod_{i=1}^{m} \mathrm{prob}_{T_i}[0]$$

We are interested in finding the set of parameters that best explain the observation, hence we seek

$$\widehat{\Theta} = \arg\max_{\Theta}\{L(\Theta, \Sigma)\}$$

In general, systems like this have multiple local minima, and optimizing over Θ is a difficult problem. For the time being we tackle this problem with a single A-Team data-flow consisting of just one memory, a single destroyer D1, and 7 construction agents called P1, P2, P3, P4, Random, Convex and Explode. This A-Team is depicted below.



D1 destroys solutions based solely on the objective function value L(Θ,Σ). P1,...,P4 construct a new solution from an old one by doing different kinds of small pertubations. Convex generates a new solution $\Theta^{new} = \lambda\Theta_1 + (1-\lambda)\Theta_2$ as a convex combination of two existing solutions. Explode counteracts the tendency of Convex to drag all parameter values toward their mean by replacing all parameters by there squared values, normalized. Random just adds a random solution to the memory.

4. Validating the model

We have no guarantee that we find the optimal set of parameters, and even if we do, our model still imposes structural restrictions on the conditional probabilities that may not hold for the A-Team. We therefore need some way of checking whether the model we have estimated for the A-Team is appropriate. The model gives us the likelihood that the sequences in our observation would lead to acceptable solutions exactly as they did. This likelihood should not be significantly smaller than what we get when we compute the likelihood for observations from a simulation of the model. There is one complication to consider. If we simulate only the set of sequences of agents that were in the real observation the likelihoods of the observations from the simulated model and the likelihood of the real observation will be directly comparable. If however the model treats the sequences in the true observation very differently than any other sequences, it is probably overtrained. To guard against this we choose to generate observations from random sequences. This is a potential problem since different sequences may have inherently different variance on how fast they transform the initial solution to an acceptable solution, but if there is nothing special about the sequences in the real observation the likelihoods should still be comparable. For the A-Team used for a job scheduling problem, described in section 7, we get the following results.

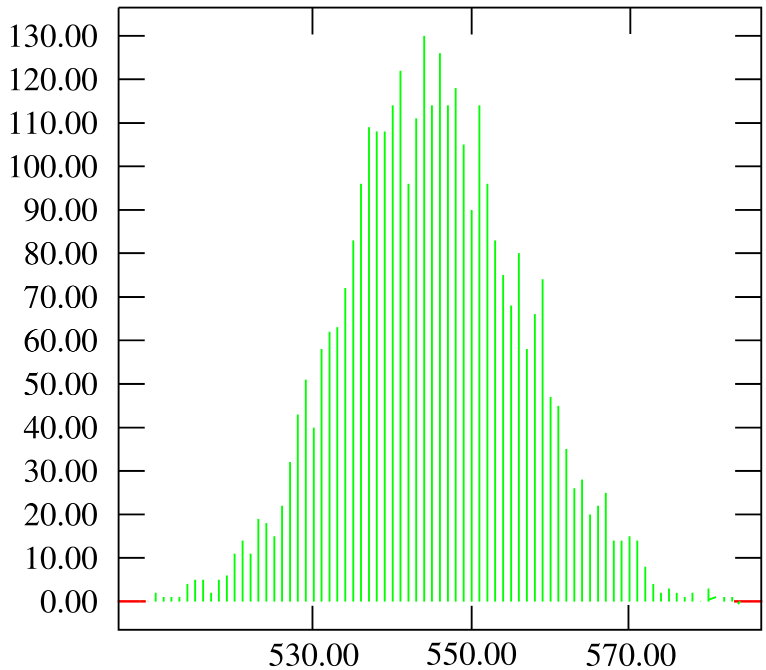
- The real observation consists of 405 sequences.

- After estimating the model we generated 3,354 simulated observations each with 405 sequences

- The negative log-likelihood of the real observation is 569
- The negative log-likelihoods of the 3,354 simulated observations range from 510 to 583.
- 2.117% of the simulated observations were less likely than our real observation.

With a regular 5% confidence level, it is unlikely that our A-team follows the model we have estimated, but the model is not too far off to be usefull in solbing the selection problem as we will see in section 8.

Figure 1. Histogram for negative log-likelihood of simulated observations



5. Doing Selection Based on the Model

The model gives us a concise description of how each agent in the A-team contributes to bringing solutions closer to the goal. It does not however directly tell us when to apply which agent. From the solution centered point of view where the decision to make is: “Which agent should work on this solution?” there are several possible answers:

1. The greedy heuristic: Choose the agent k with the highest drift = p_k - q_k. This does not take into account that the matrices $M^{k,i}$ i = 1,...,# agents; associated with applying agent k may be very detrimental to all the drifts of the next iteration.

2. The optimal choice: Choose the agent k that minimizes the expected remaining number of agent applications needed to get to the goal. This is not an easy problem to solve. We must carry around the current belief probabilities that the solution is in each S_n. Suppose for example that we have 2 agents. Agent 1 has drift p_1 - q_1 = 1 but $M^{1,1}$ and $M^{1,2}$ make the drift of both agents 0.01 in the next iteration. Agent 2 on the other hand only has a current drift of 0.5, but the drift of both agents after applying $M^{2,1}$ and $M^{2,2}$ is 1. If a solution is likely to be in S_1 then it is optimal to apply agent 1 because we will be in the goal immediately afterwards and the drift of the agents after that doesn’t matter. If we are further away from the goal it is optimal to apply agent 2. Finding the optimal selection policy is a hard problem, probably NP-hard. If our model only partially describes the true behavior of the A-team the finding the optimal selection rule given the model is probably not going to be worth the effort.

3. A reasonable approximation is to favor drift in this iteration to drift in later iterations. After all we want to get to the goal now and not later. And drift now is sure to be useful whereas drift later is only useful if the goal hasn't been reached yet at that time. A standard approach is to maximize a weighted sum of the drifts over the next few iterations. The greedy heuristic is the most trivial example of this with weights 0 for all drifts in the future.

For the time being, we only have computational results for the greedy heuristic, implemented from the solution centered perspective. Really the selection scheme should be implemented from the agent perspective: With each solution we store the p_k, q_k, r_k for each agent k, and we store the expected remaining distance to the goal, D. When agent k is going to select which solution to modify it can use it's own scheme for calculating the trade-off between working on a solution that is close to the goal, i.e. has a low D or working on a solution with p_k - q_k big in absolute terms or with p_k - q_k big compared to p_j - q_j for the other agents j not equal to k. When an agent k modifies a solution it should update D by subtracting (p_k - q_k) and it should update p_j, q_j, r_j for each agent j using $M^{k,j}$. Carrying around only the expected remaining distance to goal D rather than the entire distribution over S_1,...,S_N is an approximation, since it does not distinguish between a solution that is in S_{10} with probability 1 and a solution that is in S_1 or S_{100} with probability 10/11 and 1/11 respectively. When D becomes sufficiently negative say -10 and the solution still is unacceptable it is a sign that the solution was of the latter type and in fact it was in S_{100} so now it is probably in S_{80}, and age based destruction (see [1]) becomes useful. It remains to be seen whether carrying the whole distribution, thus allowing the agents to do precise Bayesian updating when a solution checks out to be not acceptable is worth the extra computational effort.

6. Connections to POMDP's

In this section we try to highlight the connection of our method to Partially Observable Markov Decision Processes (POMDP's).

6.1. Motivation

POMDP's have been used by several authors for robot navigation under imperfect perception. See for example [2], [3]. Our problem is very similar to that of robot navigation. A robot can take different actions, just as we can choose to apply different agents. The robot can be thought of as being in some state: its location and orientation for example. Our system is in some state: the current solution to our optimization problem. The major difference is that while a robot often receives at least partial information about which state it is in such as, there is a wall to the right of it, we only receive the very crude signal: acceptable solution / not acceptable solution. Also our state-space is huge compared to most robot environments. Nevertheless we could have modeled the A-Team with a a POMDP.

6.2. POMDP structure

in POMDP models one assumes that the set of possible states of the world can be partitioned into a finite set of classes. In each class there is a finite set of actions that can be taken. When an action is taken the state of the world changes. For each pair of classes i, j and for each possible action a there is a transition probability $p_{a,i,j}$ that if the state of the world is in class i and if action a is taken then the new state of the world will be in class j. This transition probability depends only on the class i and the action a, not on which particular state in class i the world was in, and not on the history of how it got into the particular state or class. This is the markov property. After each transition we get some observation o about the current state. For each class i there is a probability $b_{i,o}$ for observing o given we have just entered a state in class i. Finally for each class i and action a there is an instant reward R(i,a) associated with taking action a when in class i. The goal is to optimize the cumulative discounted rewards received by choosing actions. Since the class of the current state of the world is not directly observable we are forced to maintain a probability distribution over the set of classes $\pi_i = \mathrm{prob}[\text{current state is in class i}]$

There are two main tasks when using POMDP's. The first is finding the right set of classes and estimating the model parameters $p_{a,i,j}$ and $b_{i,o}$. Various methods exist for estimating $p_{a,i,j}$ and $b_{i,o}$, but none that guarantee finding the optimal solution. The second task is finding optimal actions given the model. The second task alone is NP-hard [2], even in the special case where there is only one possible observation o.

The standard approach for finding optimal actions is to compute some approximation of the value function $Q(\Pi) \to \mathfrak{R}$ that gives the expected cumulative discounted rewards given the current probability distribution over the set of classes i Π. The optimal action a^* is then given by

$$a^* = \mathrm{maxarg}_a \left\{ \sum_i \pi_i (R(i, a) + \alpha Q(\Pi(i, a))) \right\}$$

where α is the discounting factor.

6.3. A POMDP for the Agent Selection Problem

Our problem could be modeled by a POMDP. The set of states would be the set of feasible trial solutions. The set of classes would at most be the cartesian product of the set of pre-sequences and the regular partitioning $S_0,...,S_N$ of S. The reward could be 1 no matter what the action is when the class has second coordinate S_0 and 0 otherwise. The sooner we would get to a solution in S_0 the higher the cumulative discounted reward would be. The problem of course would be doing meaningful estimation of $p_{a,i,j}$ for the exponential number of classes. The standard way out would be to truncate the history, so instead of having the first coordinate of the class be the entire pre-sequence of a solution we would only remember maybe the last 4 agents in the pre-sequence. Instead we have chosen to implicitly carry around the exponentially many classes, by imposing constraints on how $p_{a,i,j}$ varies with i. This trade-off has proved usefull in practise.

The second issue is the approximation of the value function Q. Any reasonable approximation would invariably involve the expected distance to the goal. Instead of maintaining an elaborate probability distribution Π over the classes, from which to calculate the expected distance to goal in each iteration, we might as well directly store the change in expected distance to goal that will arise from running each of the agents. The expected change in distance to goal arising from applying agent k at time t in our model is exactly $p_t[k]-q_t[k]$ from the algorithm in section 3.

Of course running agent k not only gives us the immediate change in expected distance to the goal. It also modifies the ability of other agents (as well as itself) for bringing us closer to the goal in the next iteration. The direct change in expected distance to the goal in our model should be thought of as the immediate reward R(i,a).

7. A Hierarchical Planning Problem

We have applied the model for agents selection in a hierarchical planning problem for multiple machine scheduling. The planning problem briefly described is as follows: We are given a set of machines. We are given a list of jobs that must be processed by a subset of the machines in a predetermined order. For each jobs we are given the sequence of machines it must go through, an earliest availability time at the first machine and a latest due date at the last machine. We are given a machine planner that can plan jobs on a single machine. A single machine can do multiple jobs at the same time. The required time for a job to get through a particular machine depends on the entry and exit times for the other jobs being done at the same time, in a complicated fashion. Our problem is to decide on intermediate times when the job moves from one machine to the next in its sequence. The set of intermediate times constitutes the master plan. The time allotted to each machine in the sequence of each job should be such that the machine planners can find a feasible plan for each machine. For the machine planner to be able to find feasible plans for each machine it may be necessary to have the jobs finish on the last machine later than the specified latest due date. In such case a lateness is incurred. The goal is to find a master plan such that the machine planners can find nearly feasible plans for each machine, and such that the total lateness is minimized A machine plan is nearly feasible if only a few jobs are allowed to finish on one machine a few minutes later than specified in the master plan of intermediate times. It is pointless to strive for perfect feasibility in a real world setting where there are bound to be random disturbances to the system.

We solve the problem by having 6 different agents, each with its own improvement heuristic, working on a population of master plans. Since it is very time consuming to run the actual machine planner, the agents don't use the actual machine planner to check for feasibility on every machine every iteration. Instead various fast predictors are used for most machines.

8. Computational Results

To collect data for our estimation we have solved 400+ random instances of the planning problem. All instances used the same shop consisting of 6 machines, and the same number of jobs (85) over a 24 hour period. The availability and due times for each job were allowed to vary. The mean slack of each job was about 20%, meaning the time from earliest availability time to latest due time was on average 20% of the time it would take for the job to get processed if it had all machines to itself.

In each instance we started with a random master plan, and sequentially applied randomly selected agents until an acceptable solution was found. We defined a solution to be acceptable when the maximum infeasibility of a single job on a single machine was less than 300 seconds, the total infeasibility of all jobs on all machines was less than 20,000 seconds and when total lateness was less than 30,000 seconds. The collected sequences of agents that transformed the random master plans to acceptable master plans were fed to the estimation program described in section 3. The greedy selection strategy implied by the estimated model was then used on a number of new problem instances with the same characteristics. Finally random agent selection, but using the same relative frequencies for each agent as implied by the greedy selection strategy as used for comparison. The results are summarized in the figure 2-4 below.

Figure 2: Histogram for no. of agent applications required. Random selection.

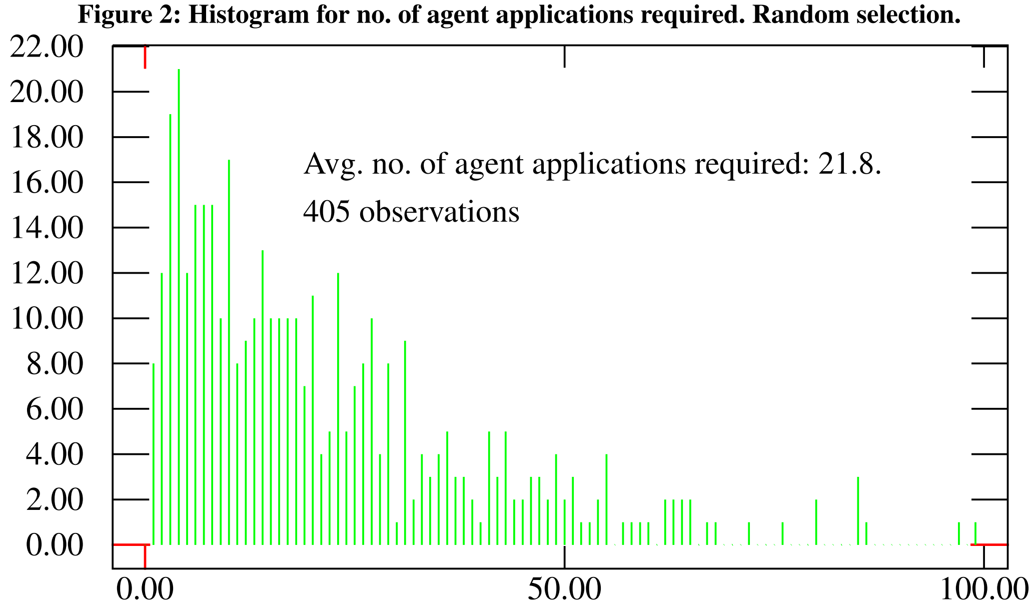


Figure 3: Histogram for no. of agent applications required. Greedy selection strategy.

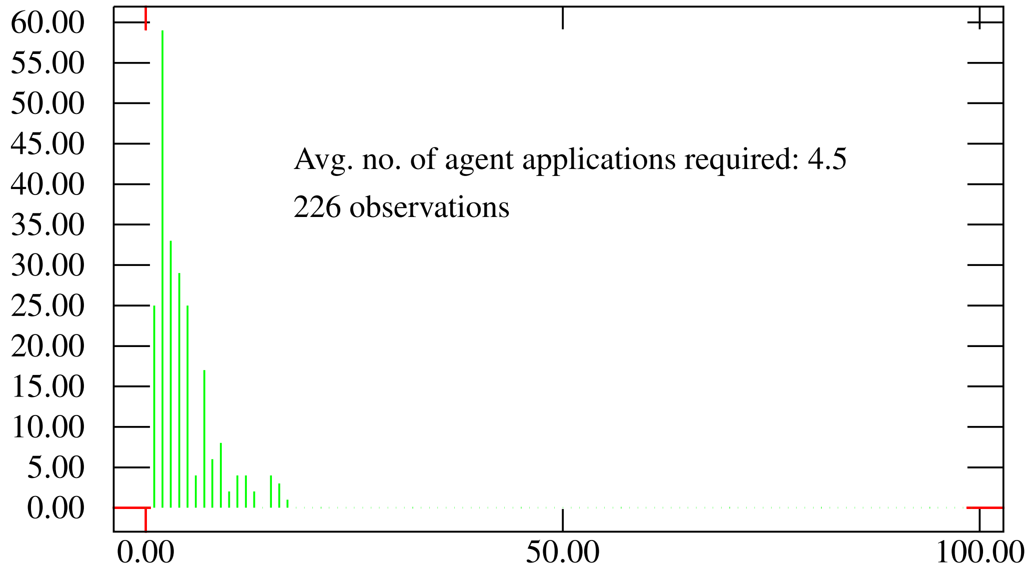
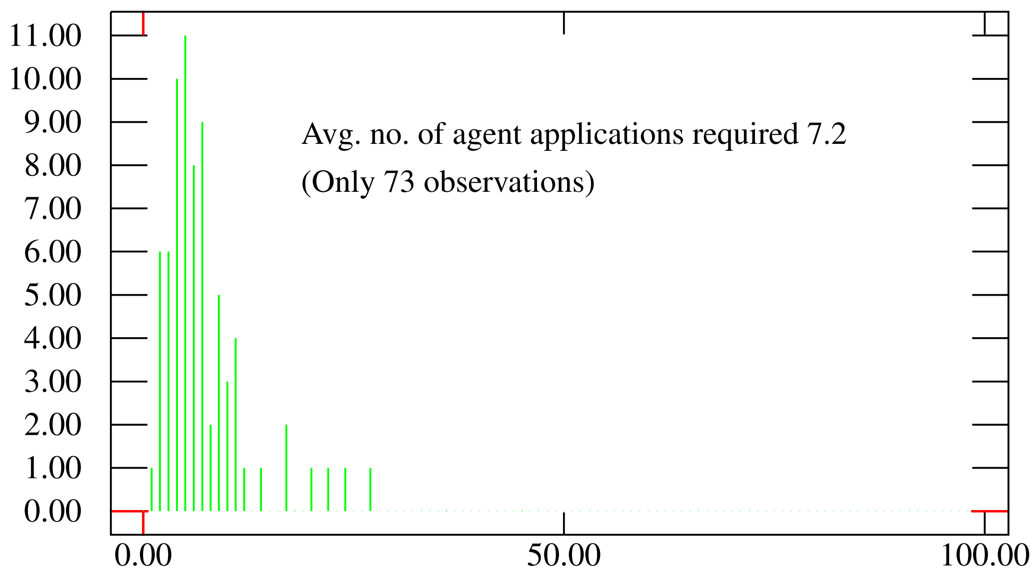


Figure 4: Histogram for no. of agent applications required. Random selection with same frequency of agents as in greedy selection.



We see that we obtain a very substantial speed-up over random selection. Even a 37% speed-up over random selection with the 'correct' frequency of each agent. This means that our model is not only able to capture the relative frequencies that each agent should run with (agent 1 should run 20% more often than agent 4 etc.), it also captures the way the agents affect each other in the $M^{i,j}$ matrices and hence is able to find sequences of agents that are substantially more effective than random sequences with the same agent mix.

9. Future work

There are many different ways to model the way the agents affect each other. There will always be a fundamental trade-off between the expressiveness of the model and the number of parameters that have to be estimated. We don't expect to have found the universally best model yet. A second issue is the A-Team for optimizing the likelihood function. Very little work has been done on that. Since we use the open A-Team architecture, it will be straight forward to add more intelligent heuristics. One obvious addition is a modification of the Baum-Welch procedure for estimating transition probabilities in POMDP's [4].

The selection strategy should be implemented from the agent perspective. We have only tested it from the solution point of view.

The current work is only a first shot at what we believe to be a very promising technique for taking advantage of historical information when an A-Team is used over and over for similar problem instances.

10. References

[1] Talukdar, Baerentzen, Gove, de Souza: “Cooperation Schemes for Autonomous Agents”, In review. email: snt@edrc.cmu.edu

[2] M. Littman: “Algorithms for Sequential Decision Making” Ph.D. thesis. http://www.cs.brown.edu/people/mll/home.html

[3] Kaelbling, Littman, Cassandra: “Planning and Acting in Partially Observable Stochastic Domains”, (unpublished) http://www.cs.brown.edu/people/lpk/home.html

[4] Rabiner, Lawrence R. 1989. “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE 77(2).

[5] Chrisman: “Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach” http://www.cs.cmu.edu/afs/cs/user/ldc/Mosaic/homepage.html