Learning Network Designs for Asynchronous Teams

Lars Baerentzen[1], Patricia Avila[2], and Sarosh N. Talukdar[1]

[1] Engineering Design Research Center
Carnegie Mellon University
Pittsburgh PA 15213
e-mail: snt@edrc.cmu.edu, lb3q@andrew.cmu.edu

[2] Power Systems Technology Division Redes de Energia S.A.
Lepanto, 350 3. 20.
08025 Barcelona, Spain
e-mail: avila@redesa.es

Abstract. An asynchronous team (A-Team) is a network of agents (workers) and memories (repositories for the results of work). It is possible to design A-Teams to be effective in solving difficult computational problems. The main design issues are: "What structure should the network have?" and "What should be the complement of agents?" In the past, the structure-issue was resolved by intuition and experiment. This paper describes a procedure by which good structures can be learned from experience. The procedure is based on the use of regular expressions for encoding the capabilities of networks.

1 Introduction

An Asynchronous Team (A-Team) is a problem solving architecture consisting of collections of agents and memories connected into a strongly cyclic directed network. The memories form the nodes of the network, the agents form the arcs. Figure 1 below shows such a network. Each memory holds a population of trial solutions. The solutions are not necessarily solutions to the overall problem that the A-Team is supposed to solve. Some memories may be dedicated to some subproblem of the overall problem, or to some relaxation of it. Each agent copies solutions from its input memory, modifies them and writes the results to its output memory. Each agent is autonomous in that it decides for itself when to work and on which solutions to work. Agents in an A-Team cooperate by working on each others results. The agents can be divided into two types: Construction agents that produce new solutions, and destruction agents that erase existing solutions from their output memories.

A construction agent consists of several parts:

- an operator, that transforms solutions from its input-memory into solutions of its out-memory type.
- a scheduler, that decides when the agent should run.

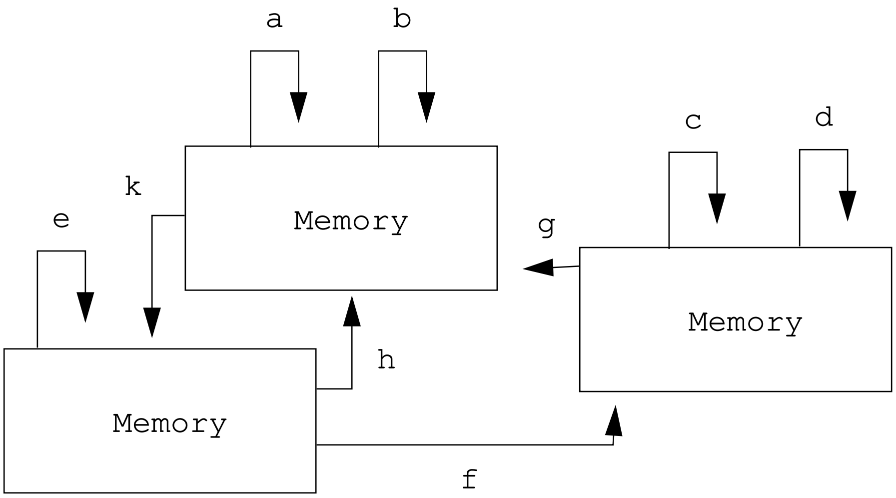

Fig. 1. An A-Team consisting of memories and agents.

- a selector, that decides which solution from the input-memory the operator should work on.

In a destruction agent the operator is missing. All the skill of a destruction agent lies in its ability to choose when to erase which solution from its output memory.

A-Teams can be very useful for solving hard problems where lots of improvement heuristics are available, but where no single heuristic performs satisfactorily on its own. In such cases good results can often be achieved by combining the different heuristics into an A-Team. Each improvement heuristic is turned into an agent by adding a scheduler and a selector.
A-Teams have been developed for a number of difficult problem-areas including sets of nonlinear equations [1], [2], traveling salesman problems [3], high-rise building design [4], reconfigurable robot design [5], diagnosis of faults in electric networks [6], control of electric networks [7], job-shop-scheduling [8], protein structure analysis [9], and train-scheduling [10], steel-mill scheduling, paper-mill scheduling and constraint satisfaction [11].

Until now A-Team builders have designed the topology of the network for an A-Team for a given problem more or less by trial and error, and by hand. In this paper we explore a technique for automatically learning good A-Team network designs. The technique relies on a method for measuring agent cooperation presented in [13]. In section 2 we first give a brief presentation of an extended version of this method. Then in section 3 we develop the concept of a Regular Expressions with Probabilities (REP) which is needed to calculate the effectiveness of different network topologies for an A-Team, and show how to use the results from section 2 to obtain good network topologies. Section 4 concludes the paper with some computational results.

2 Measuring Agent Cooperation in A-Teams

In this section we present a parametric model for how the agents in an A-Team cooperate. We then show how to use maximum likelihood estimation to estimate the parameters in the model for a real A-Team.

2.1 The model

Definition 1. For a given memory let:

- S be the space of all solutions that can be stored in memory.
- G (Goal) be the subset of S that we consider acceptable final solutions. If the A-Team can find one of these solutions we will be happy.
- S_0, S_1, S_2, ..., S_∞ 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 $S_k (k \leq n)$ for which there exists some agent that will turn the solution into a solution in S_n with positive probability.
- S_∞ be those solutions in S that are not in any S_n. In other words S_∞ are those solutions for which no finite sequence of agents has positive probability of turning the solution into an acceptable final solution.
- A solution in S_n be said to be at a distance n from the Goal.

If, given any solution, it is possible to compute which class S_n, the solution belongs to, then the underlying optimization problem that the A-Team is trying to solve can be attacked with all kinds of greedy heuristics. In this paper we are concerned with the more difficult and more realistic scenario in which we can only distinguish acceptable final solutions from non-acceptable solutions. We assume:

- There is an oracle which, given a solution, answers yes or no to the question: "Is this solution an acceptable final solution?", i.e. is this solution in S_0 or not.
- No other assesments of solution quality are available.

Whenever an agent, A, works on a solution, x, it produces a new solution $A(x)$. There are some probabilities that depend both on the solution and on the agent, that the new solution will be closer to the goal, further away from the goal, or at the same distance from goal as the old solution. To keep things simple we will assume that:

– if $x \in S_n$ then $A(x)$ will be 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 inverses.

Definition 2. The usefulness of applying an agent A_i to a solution x is the triple of probabilities (p, q, r), where

– p is the probability that the new solution, $A_i(x)$, will be one step closer to the goal than x.
– q is the probability that the new solution will be one step further away from the goal than x.
– r is the probability that the new solution will be at the same distance from the goal as x.

To model agent cooperation we define:

Definition 3. For each pair of agents A_i, A_j let

$$M^{i,j}$$

be a 3 by 3 matrix that models how the usefulness of agent A_j is changed when a solution is modified by running agent A_i. If the usefulness of running agent A_j on solution x is $\left(p_{\text{old}}, q_{\text{old}}, r_{\text{old}}\right)$ and agent A_i is used to generate solution y from solution x then the new usefulness of running agent A_j on solution y is given by

$$(p_{\text{new}}, q_{\text{new}}, r_{\text{new}}) = \left(p_{\text{old}}, q_{\text{old}}, r_{\text{old}}\right) M^{i,j} \tag{1}$$

To get a simple but complete parametric model describing how the A-Team works we define:

Definition 4. For each agent A_i let

$$(p_i, q_i, r_i)$$

be the usefulness of A_i on a randomly chosen solution from S, and

Definition 5. Let

$$\pi_0, \pi_1, \ldots, \pi_N, \pi_\infty$$

be the probabilities that a random starting solution is in S_0,S_1,…,S_N,S_∞ respectively.

The model is relatively simple, and there are of course many ways agents can cooperate that can not be captured well in this model. For instance, the situation where running agent A_i is very good after a pair of agents A_{j_1} and A_{j_2} have both been run, but not after running either of the two agents, cannot be modeled in our framework.

2.2 Estimating the Model Parameters

We assume that we have a lot of information about the A-Team available in the form of historic runs. We assume:

- We are given a number of sequences of agents, each sequence terminated by a No or a Yes. Each sequence of agents was applied to a random starting solution. As soon as an acceptable final solution was found the system was stopped and a Yes reported. If some maximum number of agents were applied before an acceptable solution was found the sequence was terminated with a No.

Now let the vector θ_0 denote a specific value of each of the parameters in the model. For a given θ_0 we want to compute the probability of reaching the goal in exactly the same manner as we did in the historical sequences.
For sequences terminated with a No, we seek the probability that the goal will not be reached after applying that particular sequence of agents. For a sequence of length n terminated with a Yes, we seek the probability that after the first $n-1$ agents have been applied the goal has not yet been reached, but when the last agent is applied the goal is reached.
The higher these probabilities are for the historical sequences, the better the historical sequences are explained by the parameter values θ_0. The better a particular θ_0 explains the historical sequences, the more likely it is, that that set of parameters is the correct set of parameters for describing the workings of the A-Team. The function that computes the probability of getting the exact observations that we have gotten as a function of the parameter values is called the likelihood function in statistics. The likelihood function in our case is rather hairy to write down. Nevertheless we have written a program (an A-Team in fact) that does a pretty good job at maximizing it. The set of parameter values that maximizes the likelihood function is known as the maximum likelihood estimate for the unknown parameters. We can show, but it is beyond the scope of this paper, that under mild conditions on how the agents used in the sequences that comprise our observations are chosen the maximum likelihood estimator will converge in probability, not necessarily to the correct underlying parameter values, but to a set of values that predict perfectly the behavior of the A-Team we are modeling.

3 Designing a Good Network Topology for an A-Team

The task of designing a good network topology consists of two major subtasks. The most important subtask is to decide on a good decomposition of the problem into related problems: subproblems, relaxed problems etc. This task is very application specific and is hard to formalize. In this section we focus on the second aspect of good topology design, which is: Given the different types of memories for the different related problems, and given that each available agent can read from and write to particular types of memories how should the network of agents and memories that compose our A-Team be configured?

3.1 Topology for doing low overhead selection

The fundamental observation about network topologies is that different network topologies allow different sequences of agents to be applied to initially seeded solutions. With the topology in fig. 2 below, all sequences of a,b and c's are possible, with the topology in fig. 3 only sequences consisting of 0 or more a's followed by exactly one b followed by 0 or more c's are possible for solutions seeded into the left memory.

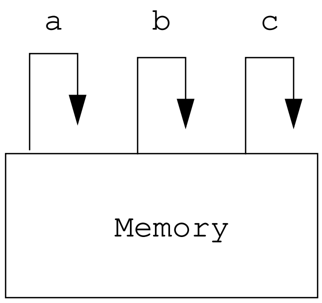

Fig. 2. All sequences containing a,b and c possible.

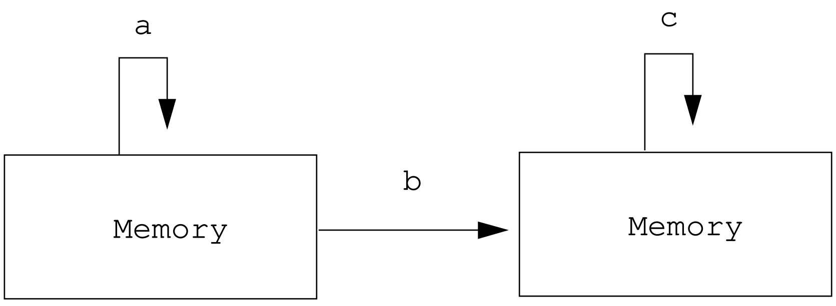

Fig. 3. Only sequences consisting of any number of a's followed by one b followed by any number of c's can be applied to solutions starting in the left memory.

Of course the individual selectors in the agents can further restrict what sequences are actually generated, but in many cases, obviously bad sequences can cheaply be avoided by having a network structure that makes them impossible.

In principle, one can always get by with only a single memory. This memory would hold the 'sum type' of each of the individual solution types, that is the memory could hold a solution of any of the types. Then it would be up to the selector within each agent to only select solutions from the memory that fit it's required input type. Of course no one would ever want to do this. We only mention this to emphasize that having multiple memories can be seen as just a way of making the selection problem for the agents easier.

3.2 Topology effectiveness.

To examine the effectiveness of different topologies for an A-Team we must fix the selection and scheduling strategies of each agent. In particular we shall assume:

- Each agent does random selection.
- Each agent is scheduled with constant frequency in each memory. Different copies of the same agent working on different memories may have different scheduling frequencies.

The result of the above assumptions is that from the solution perspective the probability that agent k will work on the solution next is some constant α_k that only depends on the relative frequency of the agents hooked up to the memory containing the solution.
Given estimates for the parameters in the model described in section 2.1 we would like to compute the expected waiting time for the earliest creation of an acceptable final solution. This we currently cannot do. We shall demonstrate however, how for a given topology one can compute the expected value of the total discounted drift towards goal:

$$E\left(\sum_{n=0}^{\infty} \gamma^n \lambda_n\right) \text{ where } 0 < \gamma < 1 \text{ is the discounting factor, and} \qquad (2)$$

λ_n is a random variable measuring the drift $(p_n - q_n)$ in iteration n.

The larger the expected discounted drift, the better the topology (and/or the better the chosen agent frequencies.) There are (at least) 2 ways to think of the discount factor λ.

- It is merely a mathematical trick to make the above sum finite, allowing us to compare two systems each having constant drift say 0.2 and 0.6 respectively per agent application. Without the discounting factor both would have drift ∞, but the latter is clearly preferable to the former.
- It makes us prefer drift that happens sooner, rather than later. This is important since the objective really is to find a final acceptable solution as fast as possible. The sooner the drift happens the sooner the acceptable final solution is found.

Regular Expressions and Finite State Automatons. To calculate the expected discounted drift to goal we need to keep track of the set of sequences of agents that can be generated by a given topology, as these are exactly the strings we are taking the expectation over. We will need the concepts of regular languages, regular expressions and finite state automatons borrowed from computer science (see for instance [14].) First some definitions:

Definition 6.

- An alphabet is a finite set of symbols $\{a_1, ..., a_n\}$. An element from the alphabet is called a letter, symbol or character.
- A string, sentence or word is a finite sequence of letters.
- A language is a set (possibly infinite) of strings.
- The concatenation of two strings x and y written xy is the sequence of letters formed by first writing down the letters that form x then the letters that form y. For example the concatenation of 'after' and 'wards' is 'afterwards'. The concatenation of two languages L and M is $LM = \{xy \mid x \text{ is a string of } L \text{ and } y \text{ is a string of } M\}$
- The language L_n is L concatenated with itself n times. L_0 is the language consisting only of the empty string.
- The union of two languages L and M is $L \cup M = \{x \mid x \text{ is a string of } L \vee x \text{ is a string of } M\}$
- The language L^* is $L_0 \cup L_1 \cup L_2 \cup \ldots$ It consists of all strings that can be made by concatenating 0 or more (only finitely many) strings from L.

We are now in position to define regular expressions and regular languages. The definitions are recursive:

Definition 7. The regular expressions over an alphabet are exactly those expressions that can be formed using the following construction rules. For each expression there is an associated regular language. The definition of the associated language is given with each construction rule.

1. The special symbol ϵ is a regular expression. It denotes the language consisting only of the empty string.
2. For each letter 'a' in the alphabet, 'a' is a regular expression denoting the language consisting of the single string 'a'
3. If R_1 and R_2 are regular expressions denoting languages L_1 and L_2 respectively then
   (a) $(R1) \mid (R2)$ is a regular expression denoting the language $L_1 \cup L_2$
   (b) $(R1)(R2)$ is a regular expression denoting $L_1 L_2$ the concatenation of L_1 and L_2
   (c) $(R1)^*$ is a regular expression denoting ${L_1}^*$

To avoid having to write all the parentheses we define an operator hierarchy. We define $*$ to have the highest priority, then concatenation and finally $|$ the union operator. We allow parenthesis to be omitted whenever it can be done without causing confusion as to which regular language is denoted.

Definition 8. A Nondeterministic Finite state Automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q, F)$ where

- Q is a finite set called the states.
- Σ is a finite set called the alphabet.
- δ is a relation $\delta \subset Q \times \Sigma \times Q$ called the possible transitions
- $q \in Q$ is a special state, called the initial state.
- $F \subset Q$ is a subset of the states called the accepting states.

One should think of an NFA as a machine that reads a string of letters from the alphabet. At any given time the machine is in one of the finitely many states in Q. The machine starts in the special initial state q and each time a letter is read the machine transitions from one state to another. The set of possible new states depend on the current state and the symbol read. If the current state is q_c and the symbol read is x then the possible new states are those q_n for which $(q_c, x, q_n) \in \delta$. The automaton is called nondeterministic because there might be more than one such q_n for a given q_c and x.

Definition 9. An NFA $(Q, \Sigma, \delta, q, F)$ is said to accept a string $x_1, x_2, \ldots, x_k$ if there exists states $q_0, q_1, \ldots, q_k$ such that

- $q_0 = q$
- $(q_{i-1}, x_i, q_i) \in \delta \;\; \forall\, 0 < i \leq k$ and
- $q_k \in F$

That is the NFA is said to accept a string if it is at all possible for the machine to be in one of the accepting states F after having read the string.
The set of strings that is accepted by a particular NFA $M = (Q, \Sigma, \delta, q, F)$ is called the language $L(M)$ accepted by M.

It is a well known result from Computer Science (see for instance [14]) that:

Theorem 10. *For any NFA M the language $L(M)$ accepted by M is regular. For any regular language L there exists an NFA M such that $L = L(M)$*

Representing Networks as Regular Expressions. Theorem 10 is very important as it allows us to look at a network of memories and agents in terms of regular expressions. Rather than calculating the total expected discounted drift to goal directly for a network of memories and agents we can translate the network to an equivalent regular expression, and calculate the total expected drift to goal based on the sequences of agents that can be produced by the regular expression.

Theorem 11. *A network of memories and agents forms an NFA*

Proof. Consider a network of memories $\{m_1, m_2, \ldots, m_n\}$ and distinct agent types $\{a_1, a_2, \ldots, a_m\}$. Define the relation $\delta \subset \{m_1, \ldots, m_n\} \times \{a_1, \ldots, a_m\} \times \{m_1, \ldots, m_n\}$ such that (m_i, a_j, m_k) whenever the network contains an arc with an agent of type a_j reading from memory m_i and writing to memory m_k. Define:

- $Q = \{m_1, m_2, \ldots, m_n\}$
- $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- $q = m_1$
- $F = Q$

Then the NFA $M = (Q, \Sigma, \delta, q, F)$ accepts exactly those sequences of agents that can be applied to solutions seeded into memory m_1

The way we are using NFA's they should not be seen as machine's accepting strings but rather as machines generating strings. The set of strings that can be generated are the same as the set of strings that the machine would accept. The machine generates a random string in the following nondeterministic way:

1. Start by setting the current state $q_c = q$, the special initial state of the NFA.
2. If the current state q_c is in F you may stop;
3. If you choose not to stop pick a random transition from $delta$ with q_c as the first coordinate $(q_c, a_{\text{new}}, q_{\text{new}})$.
4. write a_{new} to the output; set $q_c := q_{\text{new}}$; goto 2.

Including Scheduling Frequencies. There is one problem with using the standard NFA's and the standard regular expressions. They do not include information about the probability of a particular string of agents being generated. they merely represent which strings can or cannot be generated. We want a way to include information about the probability α_k of agent k working on a solution in a memory. In the case of the NFA we can easily define an NFA with probabilities (NFAP):

Definition 12. A Nondeterministic Finite state Automaton with Probabilities (NFAP) is a 5-tuple $(Q, \Sigma, \delta, q, F)$ where

- Q, Σ, q and F are as for the NFA.
- δ is a relation: $\delta \subset Q \times \Sigma \times Q \times [0;1]$ where $(q_i, a_j, q_k, p) \in \delta$ means that the transition consisting of writing a_j and moving from q_i to q_k has probability p of being applied when the machine is in state q_i

Similarly we can add probabilities to a regular expression. As for ordinary regular expressions the definition is recursive:

Definition 13. The regular expressions with probabilities (REP)'s over an alphabet are exactly those expressions that can be formed using the following construction rules. For each expression there is an associated regular language L and a probability $\nu(z)$ for each string z of the language.

1. The special symbol ϵ is a REP. It denotes the language consisting only of the empty string ϵ which has probability $\nu(\epsilon) = 1$

2. For each letter 'a' in the alphabet, 'a' is a REP denoting the language consisting of the single string 'a' with probability $\nu(a) = 1$
3. If R_1 and R_2 are REP's denoting (L_1, ν_1) and (L_2, ν_2) respectively then
   (a) $(R_1) |^p (R_2)$ is a REP denoting the language $(L_1 \cup L_2, \nu)$ where
      - If a string z is in both L_1 and L_2 then its new probability is $\nu(z) = p\nu_1(z) + (1-p)\nu_2(z)$
      - If a string z is only in L_1 then its new probability is $\nu(z) = p\nu_1(z)$
      - If a string z is only in L_2 then its new probability is $\nu(z) = (1-p)\nu_2(z)$

   (b) $(R_1)(R_2)$ is a REP denoting $(L_1 L_2, \nu)$ the concatenation of L_1 and L_2 with the probability of a string z in $L_1 L_2$ being

   $$\nu(z) = \sum_{z_1, z_2} \nu_1(z_1)\nu_2(z_2)$$

   where the sum is over those pairs of string z_1, z_2 such that $z_1 z_2 = z$, $z_1 \in L_1$ and $z_2 \in L_2$.

   (c) $(R_1)^{*(p)}$ is a REP denoting the language $({L_1}^*, \nu)$ with probabilities

   $$\nu(z) = \sum_{n=0}^{\infty} (1-p)p^n \sum_{z_1, \ldots, z_n} \nu_1(z_1)\nu_2(z_2)\ldots\nu_n(z_n) \qquad (3)$$

   where the second sum is over all sets of n strings $z_1, \ldots, z_n$ all in L_1 such that $z = z_1 z_2 \ldots z_n$

   One should think of these probabilities as occurring the following way:
      i. First flip a skewed coin with probability p of getting heads and $(1-p)$ of getting tails. If we get tails we immediately stop; other wise goto 3(c)ii
      ii. generate a random string from L_1, choosing each possible string y from L_1 with probability $\nu_1(y)$.
      iii. goto 3(c)i.

   This means that in effect we are choosing a geometrically distributed number of strings from L_1 and concatenating them. Formula (3) sums up the different ways a particular string z can be generated in this way.

The problem of converting from an NFAP to a REP and vice versa is a little complicated. There is a simple process for converting back and forth between regular expressions and ordinary NFAs. We can show, but we will not go into the proof here, that

Theorem 14. *For any REP (R, ν) there exists an NFAP $(Q, \Sigma, \delta, q, F)$, and for each NFAP there exists a REP such that*

1. *The set of strings generated by the NFAP is the same as the set of strings generated by the REP. (This we know from Theorem (10)), and*

2. *The probability that the NFAP will generate a particular string z is exactly $\nu(z)$*

Instead let us demonstrate how to calculate the expected total drift towards goal for a REP based on the agent interaction matrices from the A-Team model.

Calculating Total Expected Discounted Drift towards Goal for a REP. Notation (reminder):

- Let $\gamma < 1$ be the discounting factor.
- Let $(p_k(x), q_k(x), r_k(x))$ denote the probabilities that the new solution generated by agent k will be one step closer to goal, one step further away from goal or at the same distance from goal, respectively, compared to the solution x which the agent read in.
- Let P_x be shorthand for the entire vector

$$(p_1(x), q_1(x), r_1(x), \ldots, p_N(x), q_N(x), r_N(x))$$

  for a system with N different agents.
- Let $M^{m,n}$ be the matrix describing how running agent m changes (p_n, q_n, r_n), that is:

$$(p_n(m(x)), q_n(m(x)), r_n(m(x))) = (p_n(x), q_n(x), r_n(x))\, M^{m,n} \tag{4}$$

- let M^m be the big $3N$ by $3n$ matrix that describes how running agent m changes p,q and r for all N agents. M^m is block diagonal:

$$M^m = \begin{bmatrix} M^{m,1} & 0 & 0 & 0 \\ 0 & M^{m,2} & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & M^{m,N} \end{bmatrix} \tag{5}$$

We define

Definition 15. Let $V(R, P)$ be the total expected discounted drift towards goal from running a random sequence of agents from the language defined by the REP R given the starting solution is described by the vector P.

Note that this definition is valid even for vectors P with $p_k + q_k + r_k \neq 1$. Before we give a complete algorithm for computing $V(R, P)$ we note some simple properties and calculate some simple examples:

- Let $V(a, P)$ be the expected drift towards goal from applying agent 'a' to any solution x with $P_x = P$. Then $V(a, P) = p_a - q_a$, where p_a and q_a are the relevant coordinates of P corresponding to agent 'a'. Note that $V(a, P)$ is linear in the second argument: $V(a, \gamma P) = \gamma V(a, P)$ for any real number γ.

– Let $V(ab, R)$ be the expected total discounted drift towards goal from first apply ing agent 'a' to a solution x with $P_x = P$, and then agent 'b' to $a(x)$. Then

$$V(ab, P) = V(a, P) + \gamma V(b, PM^a) \tag{6}$$

rewriting the right hand side we get

$$V(ab, P) = V(a, P) + V(b, P(\gamma M^a)) \tag{7}$$

– It will ease the notation if we include the discount factor γ in the M^a matrix, so define for any agent m:

$$Q^m = \gamma M^m \tag{8}$$

in which case (7) becomes:

$$V(ab, P) = V(a, P) + V(b, PQ^a) \tag{9}$$

It is perhaps somewhat surprising that formula (9) has a very nice generalization to any pairs of REPs R_1 and R_2:

Theorem 16. *For any pair of REPs R_1 and R_2 there exists a $3N$ by $3N$ matrix Q^{R_1} which depends only on R_1 such that*

$$V(R_1R_2, P) = V(R_1, P) + V(R_2, PQ^{R_1}) \tag{10}$$

Proof. We give a constructive proof by showing how to recursively compute Q^{R_1}. There are five possibilities for R_1

1. $R_1 = \epsilon$
   In this case the empty sequence is the only sequence in $L(R_1)$ so P is not changed when a random sequence from $L(R_1)$ is run, hence

   $$Q^{R_1} = I$$

   Note for later use that I the identity matrix in particular is 3 by 3 block diagonal (in fact it is diagonal).
2. $R_1 = 'a'$
   In this case Q^a is as defined in (8), and hence is also 3 by 3 block diagonal.
3. R_1 is of the form $R_1 = R_{11} \mid^{\alpha} R_{12}$.
   In this case the sequence of agents run will be a sequence from R_{11} with probability α and a sequence from R_{12} with probability $1-\alpha$. Let $Q^{R_{11}}$ and $Q^{R_{12}}$ be the two matrices describing how P would be changed by running a random sequence from R_{11} and from R_{12} respectively. In the mean the new P is then

   $$P_{\text{new}} = \alpha P Q^{R_{11}} + (1-\alpha) P Q^{R_{12}}$$

   hence

   $$Q^{R_1} = \alpha Q^{R_{11}} + (1-\alpha) Q^{R_{12}}$$

   If both $Q^{R_{11}}$ and $Q^{R_{12}}$ are 3 by 3 block diagonal then so is Q^{R_1}.

4. R_1 is of the form $R_{11}R_{12}$
   In this case we first run a random sequence of agents from R_{11} changing P to
$$P_{\mathrm{new}} = PQ^{R_{11}}$$
   Then a random sequence of agents from R_{12} is run changing P_{new} to
$$P_{\mathrm{new}_{\mathrm{new}}} = P_{\mathrm{new}}Q^{R_{12}} = PQ^{R_{11}}Q^{R_{12}}$$
   hence
$$Q^{R_1} = Q^{R_{11}}Q^{R_{12}}$$
   Note that if the two matrices are 3 by 3 block diagonal, then so is their product.
5. R_1 is of the form $R_1 = R_{11}^{*(\alpha)}$
   What should $Q^{R_{11}^{*(\alpha)}}$ be? Remember that $R_{11}^{*(\alpha)}$ denotes the REP in which the string consisting of n random sequences from $L(R_{11})$ has probability $(1-\alpha)\alpha^n$ of being generated, thus we get:
$$V(R_{11}^{*(\alpha)}R_2, P) = V(R_{11}^{*(\alpha)}, P) + \sum_{n=0}^{\infty}(1-\alpha)\alpha^n V(R_2, P(Q^{R_{11}})^n)$$
   Using the fact that V is linear in the second coordinate we get:
$$V(R_{11}^{*(\alpha)}R_2, P) = V(R_{11}^{*(\alpha)}, P) + V(R_2, P(\sum_{n=0}^{\infty}(1-\alpha)\alpha^n (Q^{R_{11}})^n)) \quad (11)$$
   The infinite sum in (11) which is exactly $Q^{R_{11}^{*(\alpha)}}$ can be reduced:
$$\begin{aligned} Q^{R_{11}^{*(\alpha)}} &= \sum_{n=0}^{\infty}(1-\alpha)\alpha^n (Q^{R_{11}})^n && (12)\\ &= (1-\alpha)I + \sum_{n=1}^{\infty}(1-\alpha)\alpha^n (Q^{R_{11}})^n && (13)\\ &= (1-\alpha)I + \alpha Q^{R_{11}}\sum_{n=0}^{\infty}(1-\alpha)\alpha^n (Q^{R_{11}})^n && (14)\\ &= (1-\alpha)I + \alpha Q^{R_{11}}Q^{R_{11}^{*(\alpha)}} \Rightarrow && (15)\\ Q^{R_{11}^{*(\alpha)}} &= (1-\alpha)(I-\alpha Q^{R_{11}})^{-1} && (16)\end{aligned}$$
   where $Q^{R_{11}^{*(\alpha)}}$ only depends on $Q^{R_{11}}$ and α and can be computed using (16) by inverting $(I-\alpha Q^{R_{11}})$. Note that this is easy to do if $Q^{R_{11}}$ is 3 by 3 block diagonal since then $(I-\alpha Q^{R_{11}})$ will be so too.
   Note also, that in that case $Q^{R_{11}^{*(\alpha)}}$ will be 3 by 3 block diagonal as well since it is the inverse of a 3 by 3 block diagonal matrix

Since each of the basic forms of Q^R is 3 by 3 block diagonal, and since each of the three different ways to compute a new Q^R from the subexpressions of R maintain this property Q^R will be 3 by 3 block diagonal for any REP R.

We now show how to compute $V(R,P)$ for any REP R. Again the formula is recursive:

1. If $R = \epsilon$ then $V(R,P) = 0$ since no agents are run.
2. If $R = a$ then $V(R,P) = p_a - q_a$, where p_a and q_a are the relevant coordinates of P corresponding to agent a.
3. If $R = R_1 R_2$ then $V(R,P) = V(R_1,P) + V(R_2, PQ^{R_1})$ since after running a random sequence from R_1 starting from state P, the state of the system will be characterized by PQ^{R_1} by definition of Q^{R_1}
4. If $R = R_1 |^{\alpha} R_2$ then $V(R,P) = \alpha V(R_1,P) + (1-\alpha)V(R_2,P)$ since the sequence of agents will be chosen from R_1 with probability α.
5. It is a little more complicated to compute $V(R,P)$ when $R = R_1^{*(\alpha)}$: If we let R_1^n mean running n sequences of agents from R_1 then we get:

$$V(R_1^{*(\alpha)}, P) = \sum_{n=0}^{\infty} (1-\alpha)\alpha^n V(R_1^n, P) \tag{17}$$

Now since

$$V(R_1^n, P) = \sum_{k=1}^{n} V(R_1, P(Q^{R_1})^{k-1}) \tag{18}$$

we get:

$$V(R_1^{*(\alpha)}, P) = \sum_{n=0}^{\infty} (1-\alpha)\alpha^n (\sum_{k=0}^{n-1} V(R_1, P(Q^{R_1})^k)) \tag{19}$$

by changing the order of summation we get:

$$\begin{aligned} V(R_1^{*(\alpha)}, P) &= \sum_{k=0}^{\infty} (1-\alpha)(\sum_{n=k+1}^{\infty} \alpha^n) V(R_1, P(Q^{R_1})^k) && (20)\\ &= \sum_{k=0}^{\infty} \alpha^{k+1} V(R_1, P(Q^{R_1})^k) && (21)\\ &= V(R_1, \alpha P(\sum_{k=0}^{\infty} \alpha^k (Q^{R_1})^k)) && (22) \end{aligned}$$

Now if we use

$$\begin{aligned} X &= \sum_{k=0}^{\infty} \alpha^k (Q^{R_1})^k = I + \sum_{k=1}^{\infty} \alpha^k (Q^{R_1})^k = I + \alpha Q^{R_1} X \Rightarrow && (23)\\ X &= (I - \alpha Q^{R_1})^{-1} && (24) \end{aligned}$$

to substitute in (22) we get

$$V(R_1^{*(\alpha)}, P) = V(R_1, \alpha P(I - \alpha Q^{R_1})^{-1}) \qquad (25)$$

which is also easy to calculate given Q^{R_1}. Since Q^{R_1} is 3 by 3 block diagonal then so is $(I - \alpha Q^{R_1})$ which makes it easy to invert.

Applying the Method. To apply the method one must go through the following steps:

1. Hook up all the agents to read from and write to the same memory. Collect observations from many runs on several different instances of the problem that the A-Team is supposed to solve.
2. Use the observations gathered to estimate the parameters in the agent cooperation model.
3. Given the estimated parameters, the discounted drift for any REP can be calculated as described above. This step frees us from the rather tedious and time consuming task of actually experimenting with different network topologies for the A-Team.
4. When a REP with high drift has been identified, translate the REP to the equivalent NFAP, and hook up the agents in the corresponding network.

4 An Example

Consider the Traveling Salesman Problem, and two heuristics for solving it: Lin-Kernigan and simple heuristic, 2-city swap, which randomly swaps two cities in a tour without regard for the change in tour length. What is the best way (best network) for combining these heuristics? As humans, with our knowledge of the workings of each of these two agents, we have a feeling for the general strategy that must be used: The random 2-city swap should mainly be used for lifting LK out of its local minima. The main improvement in tour length will be done by LK. The power of the method presented in this paper is that it can uncover the optimal network for the A-Team using only the information about which sequences of agents were run, and how fast the goal was reached.

We collected data by running 500 random sequences of the two agents on the 318 city instance of the TSP problem, terminating with a Yes when a tour of length less than 2 percent above optimum is found and terminating with a No if no such tour is found after 100 agent applications.
The observations where then used to estimate the parameters in the cooperation model, we got the following maximum likelihood estimate:

```
pi_inf== 0.195782
pi[0]== 2.07479e-05
```

```
pi[1]== 0.180393
pi[2]== 0.0466464
pi[3]== 0.138674
pi[4]== 3.9718e-05
pi[5]== 0.00820783
pi[6]== 0.000126658
pi[7]== 0.214058
pi[8]== 0.0785221
pi[9]== 0.13753
p[0]== 2.1118e-23 q[0]== 1.12917e-14 r[0]== 1
p[1]== 0.0837636 q[1]== 0.915009 r[1]== 0.00122723
matrix[0][0]:
p_new== 0 p_old+ 0 q_old+ 1.00565e-05 r_old
q_new== 0.0216914 p_old+ 0.0190053 q_old+ 0.000291845 r_old
r_new== 0.978309 p_old+ 0.980995 q_old+ 0.999698 r_old
matrix[0][1]:
p_new== 3.34688e-06 p_old+ 8.06538e-05 q_old+ 3.29203e-09 r_old
q_new== 0.0080602 p_old+ 0.000816439 q_old+ 8.04717e-07 r_old
r_new== 0.991936 p_old+ 0.999103 q_old+ 0.999999 r_old
matrix[1][0]:
p_new== 0.999997 p_old+ 0.922703 q_old+ 0.59284 r_old
q_new== 2.43702e-06 p_old+ 0.0488165 q_old+ 0.397015 r_old
r_new== 9.4355e-07 p_old+ 0.0284802 q_old+ 0.0101451 r_old
matrix[1][1]:
p_new== 0.624745 p_old+ 1.15111e-06 q_old+ 0 r_old
q_new== 0.182888 p_old+ 3.22376e-05 q_old+ 7.38058e-11 r_old
r_new== 0.192367 p_old+ 0.999967 q_old+ 1 r_old
```

Where agent 0 is Lin-Kernigan and agent 1 is the random city swap. The first 11 lines gives the probabilities π_n that an random initial solution is in each of the S_n's. The next two lines gives the usefulness of each of the two agents, when applied to the initial solution. Notice that the estimates are "wrong" in the sense that we would probably expect LK to actually make progress when applied to an initial solution, and we would expect the random city swap to leave the solution at the same distance from goal, when applied to an initial solution. Instead the parameters, as estimated, state that LK will leave the solution at the same distance to goal with probability 1, and the two city swap will move the solution 1 step away from goal. This difference, between what is actually happening, and what the parameters estimate is happening, is offset by the parameters stating that the solutions start closer to the goal than they actually do.
As soon as either agent 0 or agent 1 has been run agent 1, the two city swap agent, will have usefulness $(p, q, r) = (0, 0, 1)$ as can be seen from the two rows:

```
matrix[0][1]:
```

```
...
...
r_new== 0.991936 p_old+ 0.999103 q_old+ 0.999999 r_old
..
matrix[1][1]:
...
...
r_new== 0.192367 p_old+ 0.999967 q_old+ 1 r_old
```

Running agent 0 (LK) yields a solution, that is a local optimum with respect to LK. This is captured by the estimates for matrix[0][0], which moves all probability mass to r:

```
matrix[0][0]:
p_new== 0 p_old+ 0 q_old+ 1.00565e-05 r_old
q_new== 0.0216914 p_old+ 0.0190053 q_old+ 0.000291845 r_old
r_new== 0.978309 p_old+ 0.980995 q_old+ 0.999698 r_old
```

Finally when the usefulness of LK thus is $(p, q, r) = (0, 0, 1)$ running agent 1 (2-city-swap) once changes the usefulness of LK to

$$(p, q, r) = (0, 0, 1) \begin{bmatrix} 0.9999 & 2.437\text{e-}06 & 9.435\text{e-}07 \\ 0.9227 & 0.04882 & 0.02848 \\ 0.5928 & 0.3970 & 0.01015 \end{bmatrix} \approx (0.59, 0.40, 0.01)$$

This compares well with what we found by manual inspection. If only 2 cities were swapped before LK was run again, then LK had a pretty high probability of simply undoing the swap, and finding the same local minimum as earlier. If however agent 1 was applied once more, then the new tour was sufficiently different, that LK would find a new local minimum when run again. In terms of the estimated parameters, the usefulness of LK after running agent 1 once more is

$$(p, q, r) = (0.59, 0.40, 0.01) \begin{bmatrix} 0.9999 & 2.437\text{e-}06 & 9.435\text{e-}07 \\ 0.9227 & 0.04882 & 0.02848 \\ 0.5928 & 0.3970 & 0.01015 \end{bmatrix} \approx (0.97, 0.01, 0.02)$$

We tried computing $V(R, P)$ where P is the vector of initial usefulness for each of the two agents

$$P = (2.11\text{e-}23, 1.129\text{e-}14, 1, 0.0837, 0.9150, 0.001227)$$

for three different REP's. In all cases we used discount factor $\gamma = 0.99$

1. $R = (0 \mid^{\alpha} 1)^{*(1.00)}$. This REP can be implemented in a single memory with both agent 0 and 1 reading and writing from/to it. In each iteration agent 0 is run with probability α and agent 2 with probability $1 - \alpha$. The optimal drift was found to occur when α equaled 0.36. The total discounted drift in this case was 15.1319.

2. $R = (0(1^{*(\alpha)}))^{*(1.00)}$. This REP represents repeatedly running LK once followed by a random number of 2-city swaps. It requires two memories to implement. LK reads from the first memory and writes to the second. One copy of the 2-city swap agent read from and writes to the second memory, and a second copy of the 2-city swap agent read from the second memory and writes to the first.
   The optimal drift was found to occur for $\alpha = 0.64$ which corresponds to running on average $\frac{1}{0.64} = 1.56$ applications of the 2-city swap agent between each pair of LK applications. The total discounted drift was 15.4822. The drift is slightly better than what could be achieved with a single memory.
3. $R = (011)^{*(1.00)}$ requires 3 memories to implement. LK reads from the first and writes to the second. One copy of the 2-city swap agent reads from the second and writes to the third. A second copy reads from the third and writes to the second. This set-up yields a discounted drift towards goal of 30.785. The clear superiority of this topology comes from the fact that only the optimal sequence of agents is applied. There is no randomness in what sequence is applied.

5 Future Work

The obvious next step is to devise a search strategy for automatically searching the space of REPs for the optimal REP. We still do not have a good feel for what structure there is on the Space of all REPs.
Secondly, we would like to incorporate the running time required of each agent into the model. One approach is to use a different discounting factor for each agent. Agents that take longer to run would have a smaller discounting factor than agents that run quickly.

References

1. de Souza, P., Talukdar, S.N.: Genetic Algorithms in Asynchronous Teams. Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufmann, Los Altos, CA, 1991.
2. Talukdar, S.N., Pyo, S.S., Giras, T.: Asynchronous Procedures for Parallel Processing. IEEE Trans. on PAS, Vol. PAS-102, NO 11, Nov. 1983.
3. de Souza, P.: Asynchronous Organizations for Multi-Algorithm Problems. Ph. D. dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1993.
4. Quadrel, R.W.: Asynchronous Design Environments: Architecture and Behavior. Ph. D. dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.
5. Murthy, S.: Synergy in cooperating agents: designing manipulators from task specifications. Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1992.

6. Chen, C.L.: Bayesian Nets and A-Teams for Power System Fault Diagnosis. Ph.D. dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 1992.
7. Talukdar, S.N., Ramesh, V.C.: A Parallel Global Optimization Algorithm And Its Application To The CCOPF Problem. Proceedings of the Power Industry Computer Applications Conference, Phoenix, May, 1993.
8. Chen, S.Y., Talukdar S.N., Sadeh N.M.: Job-Shop-Scheduling by a Team of Asynchronous Agents. IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambery, France, 1993.
9. Lukin, J.A., Gove, A.P., Talukdar S.N., Ho, C.: An Automated Probabilistic Method for Assigning Backbone Resonaces of (13C, 15N)-Labelled Proteins.
10. Tsen, C.K.: Solving Train Scheduling Problems Using A-Teams. Ph.D. dissertation, Electrical and Computer engineering Department, CMU, Pittsburgh, 1995.
11. Gorti, S.R., Humair, S., Sriram, R.D., Talukdar, S.N., Murthy, S.: Solving Constraint Satisfaction Problems Using A-Teams. To appear in AI-EDAM.
12. Talukdar, S., Baerentzen L., Gove, A., de Souza, P.: Cooperation Schemes for Autonomous Agents. In review.
13. Baerentzen, L., Talukdar S.N.: Improving Cooperation Among Autonomous Agents in Asynchronous Teams. Submitted to: Journal of Computational and Mathematical Organizational Theory.
14. Aho, A.V., Ullman, J.D.: Principles of Compiler Design. Addison Wesley 1977.