Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW’97, Ronneby, Sweden, May 1997, Springer-Verlag [http://www.ece.cmu.edu/afs/ece/usr/talukdar/public_html/ateams.ps].

[25] K. Carley, A. Newell, “The Nature of the Social Agent,” Journal of Mathematical Sociology, Vol. 19, No. 4, pp. 221-262, 1994.

[26] G. Bassak, “Bringing in the A-Teams,” IBM Research, No. 2, 1996

[27] S. Sachdev, “Modular Optimization,” Ph.D Proposal, Carnegie Mellon University, Feb. 1998

[14] S. N. Talukdar, V.C. Ramesh, “A Parallel Global Optimization Algorithm and its Application to the CCOPF Problem,” Proceedings of the Power Industry Computer Applications Conference, Phoenix, May, 1993.

[15] P. Avila-Abascal and S. N. Talukdar, “Cooperative Algorithms and Abductive Causal Networks for the Automatic Generation of Intelligent Substation Alarm Processors”, Proceedings of ISCAS-96.

[16] S. Y. Chen, S. N. Talukdar, N. M. Sadeh, “Job-Shop-Scheduling by a Team of Asynchronous Agents,” IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambery, France, 1993.

[17] J. Rachlin, F. Wu, S. Murthy, S. Talukdar, M. Sturzenbecker, R. Akkiraju, R. Fuhrer, A. Aggarwal, J. Yeh, R. Henry and R. Jayaraman, “Forest View: A System For Integrated Scheduling In Complex Manufacturing Domains,” IBM report, 1996.

[18] H. Lee, S. Murthy, W. Haider, D. Morse, “Primary Production Scheduling in Steelmaking Industries,” IBM report, 1995.

[19] C. K. Tsen, “Solving Train Scheduling Problems Using A-Teams,” Ph.D. dissertation, Electrical and Computer engineering Department, CMU, Pittsburgh, 1995.

[20] S. R. Gorti, S Humair, R. D. Sriram, S. Talukdar, S. Murthy, “Solving Constraint Satisfaction Problems Using A-Teams,” to appear in AI-EDAM.

[21] S. N. Talukdar, L. Baerentzen, A. Gove, P. S. deSouza, “Asynchronous Teams: Cooperation Schemes for Autonomous Agents,” to appear in the Journal of Heuristics, and visible at: http://www.ece.cmu.edu/afs/ece/usr/talukdar/heuristics.ps.

[22] K. M. Carley, “Computational and Mathematical Organization Theory: Perspectives and Directions,” Journal of Computational and Mathematical Organizational Theory, Vol. 1, No. 1, Oct. 1995

[23] L. Baerentzen and S.N. Talukdar, “Improving Cooperation Among Autonomous Agents in Asynchronous Teams,” submitted to the Journal of Computational and Mathematical Organizational Theory [http://www.ece.cmu.edu/afs/ece/usr/talukdar/public_html/likelihood.ps].

[24] L. Baerentzen, P. Avila and S.N. Talukdar, “Learning Network Designs for Asynchronous Teams,” in Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, also in

accommodating hard and fairly restrictive constraints on solution-speed.

REFERENCES

[1] G.F. Oster and E.O. Wilson, “Caste and Ecology in the Social Insects,” Princeton University Press, Princeton, NJ, 1978.

[2] A. Kerr, Jr., “Subacute Bacterial Endocardites,” Charles C. Thomas, Springfield, IL, 1955.

[3] “Handbook of Genetic Algorithms,” edited by L. Davis, Van Nostrand Reinhold, 1991

[4] H. P. Nii, “Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, Parts I and II, AI Magazine, 7:2 and 7:3, 1986.

[5] S. Kirkpatrick, C.D. Gelatt, and M.P. Cecchi, “Optimization by Simulated Annealing,” Science, Vol. 220, Number 4598, May, 1983.

[6] F. Glover, “Tabu Search-Parts I and II,” ORSA Journal of Computing, Vol. 1. No. 3, Summer 1989 and Vol. 2, No. 1, Winter 1990.

[7] S. Pugh, “Total Design,” Addison Wesley, 1990.

[8] P. S. deSouza and S.N. Talukdar, “Genetic Algorithms in Asynchronous Teams,” Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991.

[9] S. N. Talukdar, S. S. Pyo and T. Giras, “Asynchronous Procedures for Parallel Processing,” IEEE Trans. on PAS, Vol. PAS-102, NO 11, Nov. 1983.

[10] P. de Souza, “Asynchronous Organizations for Multi-Algorithm Problems,” Ph. D. Dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1993.

[11] R.W. Quadrel, “Asynchronous Design Environments: Architecture and Behavior,” Ph. D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

[12] S. Murthy, “Synergy in Cooperating Agents: Designing Manipulators from Task Specifications,” Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1992.

[13] C.L. Chen, “Bayesian Nets and A-Teams for Power System Fault Diagnosis,” Ph. D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 1992.

The main obstacles to good automatic-decision-making are:

1. The context (location and view of the system) of each decision-maker is unique. Therefore, the ideal strategy for each decision-maker is also unique.
2. The system changes with time. Therefore, the ideal strategy for each decision-maker also changes with time.
3. Many of the rules that should be included in a strategy, such as the rules for when to switch from the normal mode of control to an emergency mode of control, are not explicitly known.

Because of these obstacles, it is impractical to manually program the ideal strategy into each controller. An alternative is to provide thecontrollers with the capabilities for automatic, context-dependent and lifelong learning (by which we mean the continual transformation of experience--actual or simulated operating data--into location-specific, strategy improvements.)

Existing learning models, such as Neural Nets, Bayesian Nets and Reinforcement Learning, can deal with very large volumes of numerical data. But they cannot deal with data of high dimension, such as the state information of an entire power system. Therefore, the successful application of learning technology to a power system is contingent on decomposing the system into much smaller subsystems, each with a much smaller state-space. This decomposition can be either hierarchical or flat. But, as has been pointed out before, flat decompositions have profound advantages: they tend to be much more open and fault tolerant. Realizing these advantages will require extending the off-line technology described earlier to real-time situations, and adding learning skills to autonomous agents. If this can be done, the mechanical decision making can be taken over by autonomous agents. Each of these agents would collect only locally available data on the state of the power system and make up for its lack of a global view by collaborating with its neighbors, much as the pilots of aircraft collaborate in order to maintain a close formation. each of the agents would continually improve its performance through automatic learning.

For quasi-repetitive problems [22], that is, different instances of essentially the same general problem, it has been demonstrated that the autonomous agents in asynchronous teams for off-line problems, can learn to collaborate more effectively [23], [24]. But for real-time control, these learning techniques will have to be made more powerful and means developed for details are developed in [27].

The purpose of the grammar is to provide a means for constructing all asynchronous teams that might be used in solving any given instance of a family of off-line problems. In other words, the grammar constructively defines the space that must be searched if an asynchronous team that is good at solving the given problem-instance is to be found.

The primitives of the grammar are:

- sharable memories, each dedicated to a member of the family-of-problems, and designed to contain a population of trial-solutions to its problem.
- operators for modifying trial-solutions.
- selectors for picking trial-solutions.
- schedulers for determining when selectors and operators are to work.

Loosely speaking, the rules of the grammar are:

- Form autonomous agents by packaging an operator with a selector and a scheduler.
- Use quality-based-selection and completely parallel execution (all the agents running all the time, or as close to all the time, as the available computer resources will allow) as the default selection and scheduling strategies.
- Connect the agents and memories to form a strongly cyclic data flow.
- Compensate for construction deficiencies with skilled destruction.
- Mix agents as needed without regard to their complexity or phylla, that is, big and small software agents may be combined with humans, provided only that the humans subscribe to the communication and selection conditions prescribed for the software agents.

4. SOME RESEARCH ISSUES: REAL-TIME POWER SYSTEM CONTROL AND LEARNING

The typical power system contains thousands of distributed, mechanical decision makers (control devices), such as relays and voltage regulators. The restructuring of power systems that is now underway, will undoubtedly introduce entirely new classes of control devices, such as FACTS-controllers and automatic agents to trade in energy on behalf of customers and independent generators. What are the best strategies for these old and new controllers to employ?

long as the agent is autonomous and interacts with other agents only by modifying the trial-solutions they produce. Therefore, complex software agents can be mixed with simple software agents (which has been experimentally demonstrated [8]-[20]), and in principle, software agents can be mixed with humans (experiments are now underway to examine this part of the conjecture [27]).

2.8. Remarks

The key points made by the above analysis and arguments are:

- Construction and destruction are duals in that any effects obtained from adding construction agents can be replicated by adding suitable destruction agents.
- As construction agents with new and useful problem-solving skills are added, construction space contracts, that is, solutions of lesser quality draw closer to solutions of greater quality.
- If the drift is kept positive, then solutions of increasingly high quality become reachable as construction space contracts, that is, as new construction agents are added.
- Drift is likely to be positive if quality-based selection and perfect sequential scheduling are used.

But perfect sequential scheduling is simulated if all the agents work in parallel (which happens automatically if the agents are autonomous and enough computers are available). Also, solution-speed may be expected to increase as computers are added, at least to the extent that the times required for the work cycles of the agents are reduced by these additions. Thus, the conditions for scale-effectiveness in asynchronous teams are not overly restrictive, and one may expect scale-effectiveness to appear in many, if not most, asynchronous teams. The experimental evidence obtained so far [8]-[20] bears out this conclusion.

3. A GRAMMAR FOR ASYNCHRONOUS TEAMS

The preceding section provides a general description of the structure and properties of asynchronous teams. However, in designing these teams, it is convenient to have a more compact description. Such a description is provided by a grammar whose primitives and rules are a distillation of definitions 1-17. The main elements of this grammar are given below; the formal

$$\text{Prob}(\alpha_n) = 0.5\ \text{Max}\{\text{Prob}(\gamma_k)\}\Sigma_j\ (q_j)^2 \qquad (11)$$

when scheduling is perfect. Now, to determine the value of Prob(β_n), note that β_n can occur only if the agent that works on P_n is a destroyer and only if s* is unique. Therefore:

$$\text{Prob}(\beta_n) = \Sigma_j\ \text{Prob}(\beta_n|d_k)\ \text{Prob}\ (d_k) \qquad (12)$$

and

$$\text{Prob}(\beta_n|d_k) \leq \Sigma_j\ \text{Prob}(\mu_{kj} \cap \delta_j)$$

$$\leq \Sigma_j\ \text{Prob}(\mu_{kj})\text{Prob}(\delta_j) \qquad (13)$$

Combining (2), (4) and (13) gives:

$$\text{Prob}(\beta_n|d_k) \leq \Sigma_j\ q_{J+1-j}q_j \qquad (14)$$

Combining (6), (12) and (14) gives:

$$\text{Prob}(\beta_n) \leq 0.5\ \Sigma_j\ q_jq_{J+1-j} \qquad (15)$$

λ_n is positive if Prob(α_n), as given by (11), is larger than Prob(β_n), as given by (15), completing the proof.

Conjecture 1: Construction and destruction are duals: adept destruction agents can compensate for inept construction agents and vice versa. In other words, all the benefits of adding construction agents with new skills can also be obtained by adding destruction agents suitable skills.

As yet no experiments have been conducted to clearly demonstrate the validity of this conjecture. But a strong argument for its validity is made in [21].

Conjecture 2: The mix of agents can include humans without any deleterious consequences.

By way of justification, note that he convergence conditions do not limit the type of agent, as the most capable constructor (with the largest value of Prob (γ_k)) refrain from scheduling themselves.

Theorem 2 . In the simplified situation the drift, λ_n, is positive, if:

$$\mathrm{Max}\{\mathrm{Prob}(\gamma_k)\} > [\Sigma_j\, q_j q_{J+1-j}]/[\Sigma_j\, (q_j)^2] \tag{7}$$

In other words, sufficient conditions for λ_n to be positive are that there be at least one construction agent with a nonzero probability of being able to improve the best current solution, and furthermore, this probability must be greater than the right hand side of (7). Note that this right hand side can be considerably smaller than unity. For instance, with three solutions of quality q_1=0.1, q_2=0.2 and q_3=0.7, the right hand side of (7) is only 1/3.

Proof.

λ_n = Prob(α_n) - Prob(β_n) by definition. To determine the value of Prob(α_n), note that the events c_1, c_2,..., c_K, d_1, d_2,...,d_K are mutually exclusive, and α_n can occur only if the agent that works on P_n is a constructor. Therefore:

$$\mathrm{Prob}(\alpha_n) = \Sigma_k\, \mathrm{Prob}(\alpha_n|c_k)\ \mathrm{Prob}\ (c_k) \tag{8}$$

where Σ_k means “summation over all k.”But:

$$\mathrm{Prob}(\alpha_n|c_k) = \Sigma_j\, \mathrm{Prob}(\eta_{kj} \cap \delta_j \cap \gamma_k)$$

$$= \Sigma_j\, \mathrm{Prob}(\eta_{kj})\mathrm{Prob}(\delta_j)\mathrm{Prob}(\gamma_k) \tag{9}$$

Combining (2), (3) and (9) gives:

$$\mathrm{Prob}(\alpha_n|c_k) = \mathrm{Prob}(\gamma_k)\Sigma_j\, q_j q_j \tag{10}$$

Combining (5), (8) and (10) gives:

scheduling mechanisms, “perfect sequential scheduling.”

To define these mechanisms and analyze the simplified situation, suppose there are K constructors, K destroyers and the solutions in P_n are labelled so $q_1 \leq q_2 \leq ... \leq q_J$. Now, consider the transition from P_n to P_{n+1} and the following events:

c_k:	the transition is caused by the k-th constructor.
η_{kj}:	the k-th constructor selects the j-solution, s_j.
δ_j:	s_j is s*, the best solution in terms of distance.
γ_k:	the k-th constructor is able to improve s*.
d_k:	the transition is caused by the k-th destroyer.
μ_{kj}:	the k-th destroyer selects and erases the j-solution, s_j.

Definition 16. In quality-based selection, constructors and destroyers select randomly from the population of solutions, such that:

$$\text{Prob}(\eta_{kj}) = q_j \tag{3}$$

$$\text{Prob}(\mu_{kj}) = q_{J+1-j} \tag{4}$$

Thus, constructors are more likely to select the higher quality solutions while destroyers are more likely to select lower quality solutions (As such, quality-based selection is a symmetrical adaptation of the solution-rejection strategy used in simulated annealing.)

Definition 17. In perfect sequential scheduling, constructors and destroyers schedule themselves so that:

$$\begin{aligned} \text{Prob}(c_k) &= 0.5 \quad \text{if Prob } (\gamma_k) \geq \text{Prob } (\gamma_i) \text{ for all i not equal to k} \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{5}$$

$$\text{Prob}(d_k) = 0.5/K \tag{6}$$

Thus, a constructor is as likely as a destroyer to cause the transition from P_n to P_{n+1}. And all but

$$\lambda_n = \text{Prob}(\alpha_n) - \text{Prob}(\beta_n) \qquad (1)$$

In words, the drift at any point of the population-trajectory is the difference between two probabilities. The first is the probability that the next population will be closer to the goal, G_q, that is, the probability of one of the constructors selecting s* and improving it. The second is the probability that the next population will be further from G_q, that is, the probability that s* is unique and one of the destroyers will mistakenly erase it.

Theorem 1 . If the actions of all the agents, particularly the destroyers, are reversible, if g_0 is finite and if λ_n is positive for all n, then the expected value of N is finite. In other words, sufficient conditions for obtaining solutions of arbitrarily high quality are: a) one of the constructors must be able to replace solutions erased by the destroyers, b) at least one member of the initial population must be at a finite distance from the desired solutions, and c) the drift at all points along the population-trajectory must be positive.

Under the reversibility condition, the population-trajectory becomes a Markov chain, and the proof of the theorem follows directly from the properties of these chains. The details can be found in [21].

2.7. Drift

What affects the drift, λ_n? Some insights are obtained by considering the following simplified situation:

- The agents work in sequence and each agent modifies only one solution per work cycle. In other words, P_{n+1} is produced from P_n by a single agent.If this agent is a constructor, it adds one solution to P_n, if it is a destroyer, it erases one solution from P_n.
- The quality of each solution reflects its probability of being the best (closest) solution, that is

$$\text{Prob}(\delta_j) = q_j \qquad (2)$$

where δ_j is the event: $s_j = s^*$, and q_j is the quality of s_j, scaled so $q_1+q_2+...+ q_J=1$.

- All the agents use the same selection mechanism, “quality based selection,” and the same

Definition 14. $g(P(t), G_q)$, the separation of the set $P(t)$ from the set G_q, is the distance of the closest point in $P(t)$ to G_q, that is, $g(P(t), G_q) = \text{Min}\ \{f(p, G_q)\}$, $p \in P(t)$, where $f(p, G_q)$, the distance of any point p from the set G_q, is the minimum number of work-cycles by construction agents necessary to improve the quality of p to a level of q or better; that is, to modify p till it becomes a member of G_q.

Separation measured in this way has two noteworthy properties. First, solution-quality and separation are not directly related; the highest quality member of P(t) is not necessarily its closest member to G_q. Second, separation depends on the skills of the operators in C, the set of construction agents. To illustrate, consider two peaks (maxima) in the quality surface, s_1 and s_2. Suppose that the quality of s_1 is only slightly lower than that of s_2. Suppose that C contains only greedy, hill-climbing operators that are unable to descend from a high point in order to reach an even higher point. Then, there is no path from s_1 to s_2. In other words, the distance from s_1 to s_2 is infinite. However, if C is expanded to include operators that can go down-hill, then the distance between s_1 and s_2 will become finite. In general, as agents with new and useful problem-solving capabilities are added to C, points of lesser quality draw closer to points of greater quality.

2.6. Convergence Conditions

Let:

$t_1, t_2, \ldots$ be the discrete points in time at which P(t) changes.

$P_0, P_1, \ldots, P_N$ be a sequence (trajectory) of populations that reaches G_q in N steps, where $P_n = P(t_n)$.

$\{s_1, s_2, \ldots, s_J\}$ be the trial-solutions contained in P_n.

s^*	be the solution in P_n that is closest to G_q.
g_n	be the distance of P_n from G_q, that is, $g_n = g(P_n, G_q) = f(s^*, G_q)$
α_n	be the event: $g_{n+1} < g_n$.
β_n	be the event: $g_{n+1} > g_n$.

Prob(x) be the probability of event x.

Definition 15. The drift, λ_n at the n-th point of a population-trajectory is:

problem can be expressed as a multi-objective, constrained optimization problem, which in turn, can be approximated by a sequence of single-objective, unconstrained optimization problems.

Let:

M be the memory of a unary data flow

(S, q) be the single-objective optimization problem associated with M, where S is a set of all possible solutions to the problem, and q is a scalar measure of solution-quality. An optimal solution is a member of S that maximizes q.

G_q be the subset of S that contains only solutions of quality q or better.

P(t) be the population of trial-solutions in M at time t. (The initial value of this population, P(0), is assumed to be a randomly chosen subset of S).

C,D be the sets of construction and destruction agents that work on M, causing P to change with time.

T(q) be the expected value of t for which P(t) and G_q first develop a non-zero intersection, that is, the expected time for P(t) to evolve at least one member of quality q or better.

Definition 11. G_q is reachable if T(q) is finite, that is, if solutions of quality q or better will appear in M in an amount of time whose expected value is finite.

Definition 12. The effectiveness of M is the double: $\{q_{max}, T(q_{max})\}$, where q_{max} is the largest value of q such that G_q is reachable.

Definition 13. A unary data flow is scale-effective if adding agents increases q_{max}, and adding computers decreases $T(q_{max})$. A node in a strongly cyclic data flow is scale-effective if its equivalent unary data flow is scale-effective. An asynchronous team is scale-effective if any of its primary nodes is scale-effective. A node is a primary node if it is dedicated to CP, the problem-to-be-solved, instead of to one of its relatives.

2.5. Construction Space

Construction space, (S, g), is the space of solutions, S, together with an integer-valued function, g, that measures the separation in construction-work-cycles between subsets of S. The following definition of g is given in terms of the two subsets of principal concern: P(t) and G_q.

2.3. Families of Problems

Experience with asynchronous teams suggests that there are advantages to having them work on families of problems that include the problem-to-be-solved and some of its relatives. Perhaps progress on the easier members in such a family catalyzes progress on the more difficult ones.

Definition 10. Two problems are related if i) good solutions to one provide parts of, bounds for, or other clues to good solutions of the second, or ii) solutions of one influence solutions of the second. Two or more related problems constitute a family.

In general, a family of problems is more than a hierarchical decomposition of the problem-to-be-solved, and can include members whose only relationship is that the solutions of some influence the solutions of others. For instance, the two problems: design a car and design a process for manufacturing it, constitute a family, because the solution to the first influences the solution to the second.

The obvious way to deploy an asynchronous team on a family of problems is to dedicate each of the team's memories to one of the problems. This is achieved by arranging for the memory to hold a population of trial-solutions to its problem, and using a representation that is understood by all the agents that read from or write to that memory.

2.4. Effectiveness

An asynchronous team is started by placing a population of solutions in each of its memories. The agents then go to work on changing these populations (the constructors add new solutions while the destroyers erase old solutions). Under what conditions will good solutions appear in a population? To answer this question, we will model all strongly cyclic data flows by unary data flows, and all off-line computational problems by single objective optimization problems. The justifications are as follows. First, a node in a data flow represents several overlapping memories. These can always be lumped into a single equivalent memory and every disjoint subgraph that starts and ends at the node can always be lumped into a single equivalent agent, to yield a unary data flow. Since the "lumping" involves only a change of name and no change of structure, the dynamics of the original node are preserved by its equivalent. Second, every computational

2.2. Collaboration

Definition 6. Two autonomous cyber agents are connected if an output memory of one is an input memory of the other.

Definition 7. Two autonomous cyber agents collaborate if they are connected and they exchange data, whether the exchange is productive or not.

Any collaborative arrangement among two or more completely autonomous cyber agents can be visualized as a "data flow" (Fig. 1).

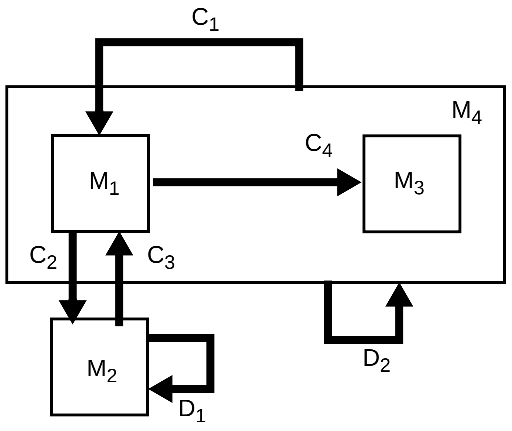

Figure 1. A data flow in which M_1, M_2, M_3, M_4 are memories, C_1, C_2, C_3, C_4 are construction agents, and D_1, D_2 are destruction agents. M_1 holds a population of trial solutions to the problem-to-be-solved. M_2 and M_3 hold populations of trial-solutions to related problems. M_4 is the union of M_1 and M_3.

Definition 8. A data flow is a directed hypergraph in which each node is a Venn diagram of overlapping input- and output-memories, and each arc represents the operator and social skills of an agent. A data flow is strongly cyclic if every one of its arcs is in a closed loop. A data flow is unary if it has one and only one node representing a single memory that is shared by all the agents in the data flow.

Definition 9. An asynchronous team is a strongly cyclic data flow, that is, a set of autonomous cyber agents, connected so their outputs can circulate, and restricted to interacting only by modifying one another's outputs.

Note that this model captures all the operating possibilities for the software agent. But it is less complete for humans in that it allows only for computer-mediated exchanges of information.

Definition 2. A cyber agent is autonomous if its control system is completely self-contained, that is, if its social skills are its only controls.

As such, an autonomous cyber agent can do what it wants when it wants. In particular, autonomous agents can choose to work in parallel all the time, if enough computers are available.

Definition 3. A cyber agent is static if it remains at the same location in its computer network, that is, if it does not switch from one set of input or output memories to another.

Thus, there are only two tasks for the control system of a static cyber agent: selection (choosing the objects to be read from its input-memories) and scheduling (determining when the operator will work on the selected objects and which of the available computers it will use for this work).

Definition 4. The work-cycle of a cyber agent consists of the following sequence: read (copy) a set of objects from its input-memories, modify these objects, and write the results to one or more of its output-memories.

In some cases, the set of objects read by a cyber agent might be empty. But the work that a cyber agent does always causes the population of objects in at least one of its output-memories to change.

Definition 5. A cyber agent is destructive if it erases objects from the populations in its output-memories. Otherwise, it is constructive.

The purpose of a destructive agent is to eliminate the mistakes and potential mistakes of its constructive counterparts by erasing outputs they shouldn't have produced and inputs they shouldn't consider.

Notice that all the intelligence of an autonomous destroyer is in its controls--its operator has only to perform the trivial task of erasing those objects its selector has chosen.

rules, of a grammar for asynchronous teams for off-line problems. Section-4 argues for the extension of this grammar to on-line (real-time) problems, particularly, distributed control and mixed initiative problems for electric power systems.

2. DEFINITIONS AND MODELS

This section develops a framework for analyzing collaborations among cyber agents, a class of agents that includes software agents as well as humans, and argues that scale-effectiveness is likely to occur in many asynchronous teams.

2.1. Cyber Agents

The environment of the any agent can be divided into two spaces--one that the agent perceives or senses (its input-space), the other, that it affects (its output-space). Of course, these spaces may overlap.

Definition 1. A cyber agent is an agent whose input- and output-spaces are maintained by computers. (Thus, both software agents and humans who happen to be engaged in computer-mediated work, qualify as cyber agents.)

Since a space is a set of objects, any computer maintained space can be thought of as a set of memories for symbolic objects. Therefore, every cyber agent in any organization of cyber agents can be modeled as:

- a set of computer-maintained memories from which the agent can read (the agent’s input space),
- a set of computer-maintained memories to which the agent can write (the agent’s output space),
- an operator (embodying the agent’s problem-solving skills) that can copy objects from the input memories, transform them, and write the results to one or more of the output memories, and
- a control system consisting of the agent’s own social skills together with any external controls, such as reporting requirements, imposed by the organization.

organizations from the primitives.

The primitives of the grammar were obtained by examining a variety of existing organizations and extracting their better features. These primitives provide for agent autonomy (as enjoyed by the members of some insect societies [1]), creation and destruction processes of adjustable strengths (as in some cellular communities [2]), populations of trial-solutions (as in genetic algorithms [3]), shared memories (as in blackboards [4]), randomized selection (as in simulated annealing [5]), lists of elements to be avoided (as in tabu search [6]), and uninhibited creation (as in brainstorming [7]).

The rules of the grammar ensure openness and ease of communication. Specifically, these rules permit only completely autonomous agents that collaborate only by modifying one another's results. Therefore, the agents have no need for external supervision; no centralized control or planning systems need be built, nor are there are any managerial layers to impede the addition or removal of agents. Moreover, the needs for commonality of expression and representation are minimal. To illustrate, consider an agent that specializes in repairing one part of the results generated by other agents. This agent needs to know only how this part is represented. It interacts with the other agents only through its repairs.

1.4. Demonstrations

One might think that agents that are autonomous and know virtually nothing about one another, would tend to work at cross purposes. Nevertheless, effective asynchronous teams have been developed for a variety of off-line problems, including nonlinear equation solving [8], [9], traveling salesman problems [10], high-rise building design [11], reconfigurable robot design [12], diagnosis of faults in electric networks [13], control of electric networks [14], [15], job-shop-scheduling [16], steel and paper mill scheduling [17], [18], train-scheduling [19], and constraint satisfaction [20]. Not only do these asynchronous teams find good solutions, but they appear to achieve scale effectiveness through fairly simple mechanisms. The succeeding material explains how and why. Specifically, Section-2 develops a framework for describing and analyzing collaborative efforts among autonomous software agents. Section-3 distills the descriptive elements of this framework into a set of primitives, and the prescriptive elements into a set of

1.2. A Design Problem

Existing software agents tend to be rich in problem-solving skills but poor in social and learning skills. Computer networks make it possible to interconnect very large numbers of software agents, and thereby, to amass enormous pools of raw problem-solving capability. How can all this capability be exploited? Of the many issues implied by this question, the subset that will be considered here is as follows.

Given:

- an instance of an off-line problem, and
- a set of software agents distributed over a computer network,

Design:

- an open, scale-effective organization for solving the off-line problem.

In other words, search through the space of all possible organizations for one whose size is easily increased and whose performance, on the given off-line problem, improves with size.

The advantages of such an organization are easy growth and fault tolerance. Costly demolition and reengineering become unnecessary for performance improvements. Instead, the problem of improving performance reduces to one of finding which agents and computers to add. Conversely, the loss of agents or computers is likely to cause a gradual rather than precipitous decrease in performance.

1.3. A Subspace Of Flat Organizations

Rather than solving the design problem above, the approach taken here is to replace it with an easier problem. Specifically, the space of all possible organizations is replaced by a much smaller subspace with a much higher concentration of open and scale-effective organizations. This subspace contains only a certain type of flat organization called an asynchronous team. These teams are always open and are often scale effective.

The subspace of asynchronous teams is defined by a constructive grammar. The two most important components of such a grammar are a set of primitives and a set of rules for composing

how it chooses to collaborate with the other information processing agents in its environment);
- and learning skills which determine how well the agent transforms its experiences into new skills.

Let:

- "an agent's environment" mean the set of all the things that the agent can affect or by which it is affected. (The environment of a software agent is mediated by a computer network and may include other types of information-processing agents, particularly, humans);
- "an off-line problem" mean a computational problem without hard constraints on solution-time, as is the case with many design, planning, scheduling, optimization and diagnosis problems. (In other words, dealing with an off-line problem involves two objectives: maximizing solution-quality and maximizing solution-speed. In contrast, dealing with an on-line or real-time problems involves only a single objective: maximizing solution-quality, but this must be done subject to deadlines, that is, hard constraints on solution-speed.)
- "collaboration" mean the exchange of data among information-processing agents, regardless of whether the exchange is productive or not.
- "an organization" mean a triple whose elements are: a set of software agents, a network of computers for the agents to use, and a prescription for the agents' collaborations. (Organizations can be of two types: hierarchic and non-hierarchic (flat). In a hierarchic organization, the prescription for collaboration relies for its effectiveness on the delegation of responsibility: the agents are arranged in layers; those in a higher layer are made responsible for, and given supervisory authority over those in the layers below. In a flat organization, there are no supervisors. Rather, the agents are autonomous and the effectiveness of their collaborations are determined solely by their social skills.)
- an "open organization" mean an organization whose size can easily be changed by the addition or removal of agents and computers.
- a "scale effective organization" mean an organization whose performance improves with size. Specifically, solution-quality improves with the addition of agents and solution-speed improves with the addition of computers.

Collaboration Rules for Autonomous Software Agents

Sarosh N. Talukdar

Carnegie Mellon University, Dept. of Electrical and Computer Engineering, Pittsburgh, PA 15213
talukdar@ece.cmu.edu

Abstract

Is effective collaboration possible among autonomous software agents that are distributed over a network of computers? Both empirical evidence and theory suggest that there are simple rules for designing problem-solving organizations in which collaboration among such agents is automatic and scale-effective (adding agents tends to improve solution-quality; adding computers tends to improve solution-speed). This paper develops some of these rules for off-line problems, and argues that the rules can be extended for the on-line (real-time) control of distributed systems, such as electric power networks.

Keywords: autonomous agents, collaboration, multi-agent systems, organizations.

1. INTRODUCTION

This paper deals with the skills that unsupervised (autonomous) software agents must have if they are to collaborate effectively. This section explains the terminology that will be used, formulates the collaboration problem and outlines an approach to its resolution.

1.1. Terminology

Let a “software agent” be any encapsulated piece of computer code, such as a program or a subroutine. Functionally, such an agent can be thought of as a bundle of skills [25] that can be divided into three categories:

- problem-solving skills which determine what the agent can do (particularly, what computational tasks it can perform);
- social or self-management skills which determine what the agent chooses to do (particularly,

The work reported here was supported in part by the National Science Foundation under Grant Number ECS-9615599, and by DARPA through Contract Number ONR Grant Number N00014-96-1-0854.