A Collaboration Strategy for Autonomous, Highly Specialized Agents[1]

S. N. Talukdar[2], S. Sachdev, and E. Camponogara[3]
Carnegie Mellon University, Pittsburgh, PA 15213

ABSTRACT

The typical planning, design or operations problem has multiple objectives and constraints. Such problems can be solved using only autonomous agents, each specializing in a small and distinct subset of the overall objectives and constraints. No centralized control is necessary. Instead, agents collaborate by observing and modifying one another's work. Convergence to good solutions for a variety of real and academic problems has been obtained by embedding a few simple rules in each agent. The paper develops these rules and illustrates their use.

Keywords: A-Teams, Agent, Asynchronous, Autonomous, Collaboration, Constraint, Cooperation, Nonlinear, Optimization, Specialized

1. INTRODUCTION

Most real world problems require that multiple conflicting objectives and constraints be satisfied. There might be tradeoffs between the different constraints that are not known a-priori and require the consultation of specialists in some aspect of the problem, but might not have any knowledge about other aspects. These specialists might be unable or unwilling to publish the algorithms they use to determine how to improve solutions with respect to their criteria. The knowledge of the experts might overlap, but there might be no expert that can solve the entire problem. Additionally, it might be difficult to synchronize these experts since they might not all be available at the same time, or might take significantly varying amounts of time to perform their tasks.

As an example of such characteristics, consider the case of spatial realization for electromechanical assemblies [6] where a set of objects is to be packed into a housing unit, and there might be many interactions between the different components. While it might be easy to find, or develop, tools to work on parts of the problem, such as improving solutions with respect to electromagnetic interference, or connectivity, or accessibility requirements for the different objects, it is usually difficult to find a single tool that solves exactly the problem under consideration. As another example, consider the case of railroad planning and scheduling [7]. Several constraints, including deadlines for pickup and delivery, availability and capacity of locomotives and cars, availability and capacity of track, must be met. Again, it is difficult to find a single tool that can solve the problem well.

There is a set of classical problem formulations which have been widely studied, and for which a range of good algorithms and heuristics exists. If the problem under consideration could be cleanly decomposed into parts that could be formulated as one of the many classical problems, the available algorithms could be applied, perhaps with little or no modification, to the subproblem. However, it is often difficult to shoe-horn the formulation of real world problems so that these existing heuristics can be directly applied. Real world problems might include aspects of several such classical problems and it is often equally difficult to extend the heuristics to take into account unique characteristics of the existing problem. Examples of such widely studied classical problems for which a large body of literature exists include the Travelling Salesman Problem (TSP) [20] and the Jobshop Scheduling Problem (JSP) [5]. An example of a real world problem embodying aspects of both these classical problems might be a factory scheduling problem where the flow of goods across the factory floor has to be taken into consideration. It is often difficult to merge heuristics to solve the overall problem. Even if the heuristics had already been combined into a single

---

1. This research was funded in part by DARPA through ONR contract N00014-96-1-0854

2. Contact person: Sarosh N. Talukdar, EDRC/ICES, Carnegie Mellon University, Pittsburgh, PA 15213. Email: snt@edrc.cmu.edu. Telephone: (412) 268-8778

3. Supported by grant #200047/96-5 of CNPq, Brazil

tool, a considerable amount of rework might be required every time there were any changes in the problem. Such changes would include the introduction of new constraints or the reformulation of existing ones.

Existing methodologies for solving multi-constraint problems can be categorized into two sets, depending on whether or not they use gradient information. There are a few widely used gradient based methods, one of the most popular being Sequential Quadratic Programming [1]. The popular non-gradient based methods include Simulated Annealing [11] and Genetic Algorithms [9].

Sequential Quadratic Programming (SQP) is an iterative method which solves a sequence of Quadratic Programming (QP) approximations to the problems, where the constraints are replaced by linearizations at the latest solution. SQP updates the values of the constraints at each iteration, and requires the ability to compute the gradient of each constraint at each iteration.

Simulated Annealing (SA) requires the computation of a merit function (which includes all constraints) at each iteration. The algorithm generates a random perturbation to the current solution at each iteration to generate a candidate solution. Then, depending on the value of the merit function before and after the perturbation, one of the current or candidate solutions is selected as the current solution for the next iteration. The selection depends on the change in the merit function and a time dependent parameter known as the “temperature”. This parameter biases the algorithm to select the better of the two solutions. The bias increases as time evolves, transforming the behavior of the SA algorithm from a random walk to a hill climb. SA does not take advantage of specialized information about the constraints, such as gradients, even if it is available.

Genetic Algorithms (GA) also require the availability to compute a merit function which is used to destroy solutions. GA works with a population of solutions, where perturbations (called “mutations”) and combinations (called “crossovers”) of existing members are used to generate new members of the population. The size of the population and its quality is controlled by culling the worst solutions at each iteration (or “generation”). GA, like SA does not make any use of specialized information about the constraints.

These methods operate in a synchronous fashion. The modules to evaluate constraint violations and gradients must be incorporated into a single structure. Combining the components may sometimes pose difficulties, for example if the constraint evaluators and modifiers are available only in executable form, perhaps because they are legacy systems or are commercially available packages for which no source code is available. The synchronization of the modules at each iteration may slow progress to the speed of the slowest component. Changes in the problem, such as the reformulation of a constraint or the addition of new constraints, require that the structure be completely reconstructed.

Asynchronous Teams (A-Teams) [13] are meta-heuristics which have been used to solve a wide variety of optimization and constraint satisfaction problems. It is based on encapsulating modules into agents which can then be organized into a team, where each agent works independently and asynchronously of each other. The agents collaborate with each other by communicating results via shared memories. A-Teams can handle multiple objectives and constraints naturally, and do not require a set of weighting functions a-priori, allowing the user to select from a set of Pareto equivalent solutions at the end of the solution process.

The focus of this work is a solution method which has been developed to allow specialist agents to cooperate. A set of agents is created, each of which is specialized to a single constraint (or objective). Each constraint therefore corresponds to a single agent, which is the only one with the ability to determine how to improve solutions with respect to that constraint. Since agents do not even have the ability to evaluate constraints other than their own, each has specialized and limited abilities. These agents can be organized into an A-Team where each works to improve solutions according to its own criterion. Since the team is modular, it is easy to modify or add constraints by replacing or adding new agents to the team. Additionally, since they work asynchronously, the agents do not have to wait for each other. The method described in this paper allows these specialist agents to cooperate effectively and find good solutions without requiring them to synchronize their activities in a lock-step fashion.

The remainder of the paper is organized as follows. Section 2 covers the structure of A-Teams and a representative sample of the behaviors their builders have observed. Section 3 develops the motivation for Asynchronous Algorithms and how they relate to their more common synchronous cousins. Section 4 provides a description and results of an A-Team approach to coordinating specialist agents to solve multi-constraint problems Finally, Section 5 summarizes the most important parts of the preceding sections.

2. WHAT IS AN A-TEAM

All the available algorithms for optimization and constraint satisfaction have weaknesses - the rigorous algorithms tend to be too slow, the heuristics, too unreliable. Rather than trying to design a new algorithm without weaknesses, a task that is difficult if not impossible, we have been working on ways to organize algorithms so that they can, suppress their weaknesses through cooperation, and together do what separately they might not. The result is a type of organization, called an Asynchronous Team (A-Team) [13], that combines features from a number of systems, particularly, Insect Societies [4], Cellular Communities [8], Genetic Algorithms [9], Blackboards [10], Simulated Annealing [11], Tabu Search (TS) [12], and Brainstorming [30].

Definition. An A-Team is a set of autonomous agents and a set of memories, interconnected to form a strongly cyclic network, that is, a network in which every agent, or almost every agent, is in a closed loop.

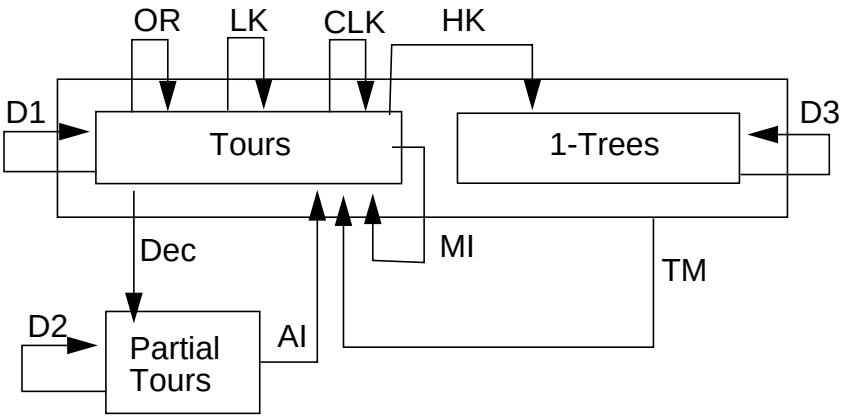

Figure 1. An A-Team to solve the Travelling Salesman Problem

An A-Team can be visualized as a directed hypergraph, called a data flow, like that in Figure 1, an A-Team to solve the TSP. Each node represents a complex of overlapping memories, which may contain solutions to the problem (tours), partial solutions (partial tours), or solutions to related problems (1-trees) that can be used to generate new solutions to the main problem. Each arc represents an autonomous agent. Results or trial-solutions accumulate in the memories (just as they do in Blackboards) to form populations (like those in GA). These populations are time varying: new members are continually added by construction agents (OR, LK, CLK, HK, MI, AI, Dec, TM), while older members are being erased by destruction agents (D1, D2, D3). More specifically, a construction agent copies results from the population at its tail, modifies the copies and inserts them into the population at its head. A destruction agent examines a population and erases those results it feels should not be there, often working from a list of results to be avoided (like the lists used in TS). The construction agents use a variety of algorithms to generate solutions, for example LK uses the Lin-Kernighan algorithm [19]. Destruction agents remove bad or unpromising solutions. Details can be found in [13].

The numbers of construction and destruction agents can be arbitrarily large and each agent, whatever its type, can be arbitrarily complex. Consequently, the problem-solving skills of a data flow can be arbitrarily apportioned between construction and destruction. Other synthetic problem-solving systems invariably concentrate on one or the other. Hill climbing, for instance, concentrates on how to construct new and better solutions while SA, GA and TS concentrate on how to destroy or reject weak solutions. Natural systems, however, often benefit from a more symmetric use of construction and destruction. The process of lamellar bone growth [16], for instance, relies as much for its efficacy on cells that add bone material to surfaces where the stress is high, as it does on cells that remove bone material from surfaces where the stress is low.

All the agents in an A-Team are autonomous. An autonomous agent decides for itself what it is going to do and when it is going to do it (like the adult members of insect societies). Consequently, there can be no centralized control. But new autonomous agents can be easily added (there is no centralized system to get in their way).

Agents cooperate by working on one another's results. Because the agents are autonomous, this cooperation is asynchronous (no agent can be forced to wait for results from another). Rather, all the agents can, if they so choose, work in parallel all the time. (Other synthetic problem-solving systems often include precedence constraints to force at least a partial order on the activities of their computational modules. Traditional Genetic Algorithms, for instance, require destruction to cease while construction is in progress, and vice-versa.)

2.1. The Effects of Scale

One might think that A-Teams would be prone to anarchy. After all, every agent does what it wants when it wants, and makes its decisions without knowing anything about the other agents except for the results they produce. Nevertheless, useful A-Teams have been developed for a wide variety of optimization and constraint satisfaction problems, including, nonlinear equation solving [14], [26], traveling salesman problems [21], high-rise building design [15], reconfigurable robot design [16], diagnosis of faults in electric networks [17], control of electric networks [18], [27], job shop scheduling [23], steel and paper mill scheduling [28], [29], train scheduling [22], and constraint satisfaction [25]. Not only do these A-Teams produce very good solutions, but they appear to be scale-effective.

Definition. An organization is scale-effective if its performance improves with size. A computer-based organization is scale-effective if there are agents whose addition improves solution-quality and computers whose addition improves solution-speed.

Scale-effectiveness is an extremely desirable property. The problem of improving the performance of a scale-effective organization reduces to one of finding which components to add. A non-scale-effective organization faces the much more difficult problem of finding which of its parts to eliminate or modify before additions can be of benefit. Synthetic organizations are often non-scale-effective, being described by the proverb, "too many cooks spoil the broth." That is, there comes a point when the addition of another "cook," no matter how competent she may be, has a negative impact on overall performance. In a scale effective organization there can never be too many "cooks."

3. ASYNCHRONOUS ALGORITHMS

The domain of Nonlinear Programming (NLP) problems was selected to demonstrate the feasibility of using an A-Team of highly specialized agents to solve constrained optimization problems. These problems usually have a single objective and multiple constraints; any or all of which may be nonlinear. The form of the problem is

$$\min f(x)$$

$$\text{s.t. } c_i(x) <= 0$$

where $f(x)$ is the objective function, and $c_i(x)$ is a set of constraints.

3.1. A Synchronous Algorithm

Sequential Quadratic Programming is an extremely popular algorithm for solving NLP problems, and is a good example of a synchronous iterative algorithm. SQP consists of generating a new iterate by creating a quadratic programming (QP) approximation to the problem at the current iterate. The solution of the QP provides a direction along which a line search is performed to find an improving iterate. The latest function values, gradients, and hessians of the objective and constraints are always used in generating the QP and to perform the line search.

3.2. The Asynchronous Algorithm

An asynchronous algorithm was developed to coordinate the efforts of a set of independent specialist agents. The specialization is represented in NLP problems by mapping each constraint (or the objective) to exactly one agent. The agent is not provided with the knowledge to compute other constraints. Agents operate asynchronously in an A-Team, and can only communicate by means of shared memories. Therefore, they cannot use the abilities of others while they are working on a solution. However, this asynchronous behavior provides the advantage that a set of evaluation agents can be easily added to the team to rapidly update, in parallel, the evaluations for solutions. The method consists of an algorithm that is followed by each agent that generates new values of the iterate.

In addition to the current iterate, the solution contains the latest available evaluations, a small number of the most recent iterates (the history of the solution), and directions recommended by other agents. This allows agents to communicate a limited amount of information about their constraints to others (by the recommendations), and for agents to infer trends in the solutions trajectory through the decision space (by the history).

The method can be described by considering the case of a single agent, lets say the agent which specializes in improving solutions with respect to a particular constraint. The agent starts by selecting a solution which violates this constraint to work on. It computes a preferred, or improving, direction in decision space with respect to its constraint. This is the ability that makes the agent a specialist. This preferred direction is modified by taking into account the recommendations of other agents. A simple process to perform this modification keeps overhead small. The agent classifies the recommendations of other agents into three groups: those it will ignore, those it will follow, and those it will attempt to stay orthogonal to. A trend is determined from the history of the solution. The recommendations, trend and a small amount of randomization are used to modify the preferred direction to generate a direction to be followed. The agent uses this and a step size determined from the history to generate a new iterate. The algorithm is described in Figure 2.

The steps followed by the agent specializing in a constraint:

Step 1: Select a solution from the input memory for which the constraint is violated

Step 2: Compute the preferred direction of the agent.

Step 3: Classify the recommendations of other agents into three groups; those which will be ignored, those which will be followed, and those which the agent will try not to oppose.

Step 4: Use the recent history (recent points visited in the solutions trajectory) to determine trends.

Step 5: Use the preferred direction, recommendations and trend information to determine a direction in decision space.

Step 6: Use the recommendations or history to determine a step size and take a step.

Step 7: Update the values of the iterate, the recommendation and the history. Write the solution to the output memory.

Figure 2. The Asynchronous Algorithm (AA)

Since agents must operate independently and asynchronously, the possibility of having a lock-step process with each agent providing input at each iteration is precluded. In order for a set of such agents to cooperate, they must communicate with each other, but the asynchronicity requirements limit the type of such communications. Agents communicate by marking solutions with additional information that is written to shared memories.

In the case of the demonstration problems, agents were allowed to add a recommendation for direction, as well as a time-stamp for when the recommendation was made. The time-stamp on recommendations allowed agents to determine how much importance to place on them, since old recommendations might be invalid and hence best ignored. An agent updated its recommendation every time it operated on a solution. The experiments assumed that evaluations are usually much easier to compute than the gradients, and that the agents worked with current solutions that had all the evaluations updated by evaluator agents (which were fast and worked in parallel).

4. DEMONSTRATION / RESULTS

A representative set of standard benchmarks for nonlinear programming codes was selected to test the method. Table 1 below is a brief description of the problem characteristics. Problems 4, 8, and 9 had equality constraints which were converted to a pair of inequality constraints. A small total constraint violation of 1e-5 was permitted in order to avoid numerical problems. The table lists the total number of constraints as used to benchmark the problems (equality constraints are counted twice). The

type of the problem is an indication of the original source of the problem. Problems for which the objective and constraints have first and second derivatives in the entire feasible region are considered regular.

Table 1. Description of test problems

#	source, problem id.	type	objective	constraints	regularity	variables	constraints
1	[3] 2.3	theoretical	quadratic	linear	regular	13	9
2	[3] 3.1, [2]106	practical	linear	quadratic	regular	8	6
3	[2], 100	practical	polynomial	polynomial	regular	7	4
4	[2], 80	practical	general	polynomial	regular	5	6
5	[2], 113	practical	quadratic	quadratic	regular	10	8
6	[2], 117	practical	polynomial	quadratic	regular	15	5
7	[2], 56	theoretical	polynomial	general	regular	7	8
8	[2], 107	practical	polynomial	general	regular	9	12
9	[2], 85	practical	general	general	irregular	5	35
10	[2], 67	practical	general	general	irregular	3	14

Each constraint was assigned to an agent, with a separate agent for the objective. Each agent had the ability to determine improving directions in the decision space with respect to its own constraint (i.e. the ability to compute the gradient of c_i). Agents were not provided with the knowledge required to evaluate or compute the gradients of other constraints.

The problems were tested with several possible implementations of steps 3 to 6 of the algorithm which compute recommendations, trend and step size. Details of these implementations are provided in the appendix. The main parameters were also varied to determine a good set of values, as well as to obtain an indication of the robustness of the method. Table 2 summarizes the test results.

The result values reported are averages over all ten problems. The mean number of iterations is the mean iteration count at which the best solution was found. Each iteration corresponds to one gradient computation (of one of the constraints, or the objective function). The mean error is the deviation of the objective value of this solution from the reported optimum.

Table 2. Summary of experimental results for the Asynchronous Algorithm

implementation of algorithm	parameters		mean for problem set	
	p_r	p_t	# of iterations	error%
AA_1	0.35	0.7	7365.7	0.706568
	0.7	0.35	7945.1	0.499582
	0.7	0.7	8114.8	0.579164
AA_2	0.35	0.7	7587.3	0.981708
	0.7	0.35	8100.9	0.436826
	0.7	0.7	7801.8	0.346626
AA_3	0.35	0.35	8034.5	0.574749
	0.35	0.7	8318.7	0.535977
AA_4	0.35	0.35	7828.6	0.72981
	0.35	0.7	7867	0.185027

Experimental results for several possible implementations of the algorithm produce similar results, and find good solutions over a range of the parameters. This indicates that it is not difficult to devise a robust implementation of the algorithm that can find good solutions.

5. CONCLUSIONS

There are many real world problems that require the satisfaction of multiple objectives and constraints. It is often difficult to modify existing algorithms or develop new ones to tackle such problems. It is possible to solve such problems using a set of specialized autonomous agents, which may use existing algorithms to solve subproblems, organized in an Asynchronous Team (A-Team).

A general Asynchronous Algorithm was proposed to convert specialized solvers into specialist agents, and to enable them to cooperate effectively to find good solutions. Implementations of this algorithm were devised to solve Nonlinear Programming problems. These were tested with a set of representative benchmark problems of theoretical and practical interest. The algorithm found near optimal solutions, and experiments in varying parts of the algorithm indicated its robustness.

One should note that the A-Team paradigm is extremely flexible, and there are many different ways of creating A-Teams to solve such problems. Current efforts are in progress to develop A-Teams of specialist agents to solve problems of significant practical importance, specifically in the domain of spatial realization for electromechanical engineering artifacts. The components of such artifacts may have non-convex shapes and complex interactions with each other, for example in connectivity, interference, separation and accessibility requirements.

ACKNOWLEDGMENTS

We would like to thank Lars Baerentzen for helpful discussions and comments.

REFERENCES

1. P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, San Diego, 1981.

2. W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Springer-Verlag, 1981.

3. C.A. Floudas and P.M. Pardalos, *A Collection of Test Problems for Constrained Global Optimization Problems*, Springer-Verlag, 1990.

4. G.F. Oster and E.O. Wilson, *Caste and Ecology in the Social Insects*, Princeton University Press, Princeton, NJ, 1978.

5. K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, NY, 1974.

6. M. C. Lin and D. N. Manocha (editors), *Applied computational geometry: Towards Geometric Engineering*, ACM Workshop on Applied Computational Geometry 1996, selected papers, Springer, NY, 1996.

7. J. H. Armstrong, *The Railroad: What It Is, What It Does: The Introduction to Railroading* (3rd edition), Simmons-Boardman Books, Inc., Omaha NE, 1993.

8. A. Kerr, Jr., "Subacute Bacterial Endocardites," Charles C. Thomas, Springfield, IL, 1955.

9. L. Davis (editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

10. 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.

11. S. Kirkpatrick, C.D. Gelatt, and M.P. Cecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, Number 4598, May, 1983.

12. 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.

13. S. Talukdar, L. Baerentzen, A. Gove, P. de Souza, “Asynchronous Teams: Cooperation Schemes for Autonomous Agents”, to appear in *Journal of Heuristics*, 1997.

14. P.S. de Souza and S.N. Talukdar, “Genetic Algorithms in Asynchronous Teams,” *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA, 1991.

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

16. 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.

17. 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.

18. 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.

19. S. Lin and B.W. Kernighan, “An Effective Heuristic Algorithm for the Traveling-Salesman Problem,” *Operations Research*, Vol. 21, 1973, pp. 498-516.

20. M. Held and R.M. Karp, “The Traveling-Salesman Problem and Minimum Spanning Trees,” *Operations Research*, Vol. 18, 1138-1162, 1970.

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

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

23. 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.

24. J. A. Lukin, A. P. Gove, S. N. Talukdar and C. Ho, “Automated Probabilistic Method for Assigning Backbone Resonances of (^{13}C, ^{15}N)-Labelled Proteins,” *Journal of Biomolecular NMR*, 9, 1997.

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

26. 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.

27. 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*.

28. 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.

29. H. Lee, S. Murthy, W. Haider, D. Morse, “Primary Production Scheduling at Steel making Industries,” IBM report, 1995

30. S. Pugh, *Total Design*, Addison Wesley, 1990

APPENDIX

This appendix provides details for the solution representation, as well as for four alternative implementations (AA_1-AA_4) for Nonlinear Programming problems of the Asynchronous Algorithm (AA) described in Figure 2.

Solution Representation

A solution consists of: { x^k, E, R, H, k}

where

x^k : the value of the current iterate

$E = \{c_0, c_1 \ldots c_n\}$: the latest available evaluations (here c_0 is used in place of f to simplify the notation)

$R = \{(r_i, k_i)\}$: a list of preferred directions of other agents (r_i), accompanied by a time stamp k_i which is the iteration at which this direction was last updated

$H = \{x^{k-1}, x^{k-2}, \ldots, x^{k-W}\}$: the history of the current iterate, consisting of a list of the W most recent iterates. (The results provided below all had W=10)

k : the current iteration

Implementation (AA_1)

The steps followed by the agent specializing in constraint i are:

Step 1: Select a solution from the memory for which the constraint c_i is violated

Step 2: Compute the preferred direction of the agent. For the NLP problems, use the gradient to determine this.

$r_i = -\nabla c_i(x^k) / | \nabla c_i(x^k) |$

update R with the current value r_i and k_i

Step 3: Classify the remaining components of R into three categories R_{ignore}, R_{follow}, $R_{orthogonal}$. Ignore old recommendations, those for which the recommendations are almost orthogonal to the preferred direction, or those for which the constraint seems satisfied. Go orthogonal to those recommendations that oppose the preferred direction, or for which the constraint is near its active point. Follow the rest.

$R_{ignore} = \forall\, r_j \in R$ s.t. ($k_j < k - 200$) or ($85^\circ < \angle(r_i, r_j) < 95^\circ$) or ($c_j <$ -1e-5)

$R_{orthogonal} = \forall\, r_j \in R \backslash R_{ignore}$ s.t. ($\angle(r_i,r_j) > 95^\circ$) or ($|c_j| <$ 1e-5)

$R_{follow} = R \backslash (R_{ignore} \cup R_{orthogonal})$

Compute recommend, a vector combining all the recommendations of other agents that will be followed.

recommend = $\sum [r_m], \forall\, r_m \in R_{follow}$

Step 4: Combine H to obtain trend, a vector capturing trends in the trajectory of the solution through the decision space.

trend = $\sum [(x^k - x^{k-m})/m], (1 \le m \le W), x^{k-m} \in H$

Step 5: Compute a direction in which the agent will take a step. This consists of the preferred direction perturbed by recommendations of other agents, trends in the trajectory of the solution, and a small amount of randomization. The amount of perturbation is determined by two parameters which are experimentally determined. Since all the vectors in the computation of direction' below are normalized, reasonable values of these parameters would be on the order of 1.

direction' = r_i + random + p_r * recommend + p_t * trend
where random is a small random vector (e.g. length ≤ 0.1) and p_r and p_t are experimentally determined parameters

proj = projection of direction' into subspace spanned by $R_{orthogonal}$

direction = (direction' - proj)

(if direction = 0, which occurs very rarely, use direction = direction')

normalize direction

Step 6: Determine a step size along direction to find x^{k+1}

stepsize = 1.5 * | trend |

to facilitate convergence a limit (that decreases as k increases) may be imposed on the stepsize.
max stepsize = q * $e^{-0.001* k}$, where q is set to a small fraction of the smallest allowed range of the variables, to reflect the scale of the problem

$x^{k+1} = x^k$ + stepsize * direction

Step 7: Update H by adding x^k and removing x^{k-W}

Replace x^k with x^{k+1}, and k with k+1 in the solution

The agent for the objective uses the same algorithm as the agents for the constraints except for the following. Since there is no target "threshold" value for the objective, the agent selects solutions that it has not worked on recently. Also, if the constraint violations are large the agent will set the new iterate to the old value, skipping steps 3 to 8.

Alternative Implementations

In order to check the sensitivity of the method to the rules used to compute trend, recommend and stepsize, the following alternative implementations were tested.

AA_2: Same as AA_1, except that recommendations are discounted by their age

recommend = $\sum [r_m * 0.95^{k_m - k}]$, $\quad \forall\ r_m \in R_{follow}$

AA_3: Same as AA_1, except that trends are computed from the directions used in the past discounted by age and the stepsize depends on the amount of perturbation made to the preferred direction.

trend = $(x^k - x^{k-1}) / | (x^k - x^{k-1}) | + \sum 0.98^m * [(x^{k-m} - x^{k-m-1}) / | (x^{k-m} - x^{k-m-1}) |]$, $\quad (1 \le m < W)$

stepsize = $(0.9 + 0.7 * r_i$*direction) * $(x^k - x^{k-1}) / | (x^k - x^{k-1}) |$

AA_4: Same as AA_3, except that recommendations are discounted by their age

recommend = $\sum [r_m * 0.95^{k_m - k}]$, $\quad \forall\ r_m \in R_{follow}$