# A Computational Model for Portfolios of Cooperative Heterogeneous Algorithms for Discrete Optimization

**Eugene Santos Jr.**
Department of Computer Science & Engineering
University of Connecticut
Storrs, CT 06269-3155
eugene@cse.uconn.edu

## Abstract

Discrete optimization problems arise throughout many real world domains including planning, decision making, and search. NP-hard in general, these problems require novel approaches to algorithm design and development. Algorithm portfolios which consist of multiple algorithm instances and types executing in parallel has been a key novel approach to these problems. We focus on cooperative portfolios where algorithms actively communicate and interact with one another during problem solving to more efficiently determine the optimal solution. Unfortunately, the added dimension of cooperation greatly complicates our understanding of the method. The goal of this paper is to provide the first comprehensive formal computational model of portfolios of cooperative heterogeneous algorithms in order to lay the foundations for rigorous analyses of these portfolios in hopes of eventually leading to provable guarantees of performance, effectiveness, and efficiency.

## Introduction

This paper addresses a fundamental problem in discrete optimization, namely the development of algorithms that maximize solution quality within a given time limit. We focus here on collections of cooperating algorithms as the basis for improving solution quality. Our goal is to provide a formal model that captures their overall behaviour which can then be used to design cooperative solutions predicated on rigorous analysis and performance guarantees.

Discrete optimization problems such as manufacturing scheduling (Luh 1997), probabilistic reasoning (Santos & Santos 1987; Shimony 1994; Santos 1994), protein folding (Neumaier 1997; Santos & Santos 2000), and cryptography (Menezes, van Oorschot, & Vanstone 1997) are in general NP-hard (Garey & Johnson 1979). The combinatorics inherent in the space of possible solutions for these problems often requires exponential computation time with respect to problem size.

In practice, this has often resulted in attempting to construct specialized algorithms tailored specifically to solve a given problem instance or highly restrictive subclass. Typically, such tailoring involves extensive problem analysis, restructuring, and simplification in order to develop these search strategies/heuristics not to mention the continual "tweaking" afterwards of the algorithm.

In reality, unfortunately, even seemingly small perturbations/variations on a known problem instance can result in huge computational differences and costs.[1] Still, for all these efforts, we have been able to identify different classes of algorithms that work well in different ways on different problem sets. A great variance exists in individual algorithm performances over different problem instances. In fact, it is often the case that an algorithm that performs best for one problem instance, may perform much worse than another algorithm on another problem instance. However, matching the best algorithm to a problem instance can be as difficult as determining the correct solution to the problem itself. Although some indication on how to do that may be available from an inspection of the problem, such predictions are notoriously unreliable. Further complicating matters is the possibility that no single algorithm currently exists which can solve the given problem in a reasonable amount of time and space.

One avenue of research that has been explored to avoid the problems of algorithm tailoring and brute force problem pre-analysis is algorithm portfolios (Gomes & Selman 1997; Huberman, Lukose, & Hogg 1997). Multiple algorithms are selected to form a portfolio. These can be multiple instances of a single randomized algorithm to multiple types of algorithms. All algorithms are run in parallel in hopes of getting good performance over a wider range of problems. The choice of algorithm portfolios is akin to how one chooses stock portfolios where a balance among available resources versus expected return is managed according to acceptable risk. However, algorithms in these portfolios typically do not interact or cooperate but it has been noted

[1]The theoretical study of "phase transitions" and heavy-tail distributions attempts to identify when a problem instance is likely to be "computationally tractable" (Hogg, Huberman, & Williams 1996; Motwani & Raghavan 1995; Gomes & Selman 1997).

that such cooperation should be beneficial (Hogg 2000).

Santos et al have been studying cooperative algorithms since the early-mid 90s (Santos 1993; Williams, Santos, & Shimony 1997; Santos, Shimony, & Williams 1995; 1999). In particular, they focused on deploying parallel multiple interacting algorithms to solve problems in Bayesian Networks (Pearl 1988). They constructed various algorithm types such as genetic algorithms, best first search, and integer programming. Using the Overmind Architecture (Williams, Santos, & Shimony 1997), they conducted an extensive number of experiments to determine the effectiveness of their approach. (Santos, Shimony, & Williams 1999) demonstrated that by changing the mix of a portfolio on-line based on observing the current and past performance of the portfolio leads to significant speed-ups. Others have also recently explored a similar approach (Denzinger & Offerman 1999; Baerentzen, Avila, & Talukdar 1997; Talukdar *et al.* 1998).

The immediate benefit seen in the experiments from a portfolio approach is the fact that: Given multiple algorithms solving the same problem, the elapsed time needed to find the solution is in general bounded above by the best single algorithm should it have been run individually. This is the basic strength of portfolio approach but is not all that surprising.

Of more interest are the situations in which cooperation between algorithms indeed led to faster discovery of the optimal solution. For example, empirical results demonstrated that genetic algorithms are good at finding good local solutions quickly. Hence, the speed of the genetic algorithms effectively jump started other slower algorithms with better "seed" candidates. Furthermore, combining exact algorithms with randomized algorithms provides guarantees of optimality. It is well known that genetic algorithms do not guarantee optimality in their final solutions.

However, the most convincing argument for cooperative portfolios is the identification of problems where no single existing algorithm is capable of solving the problem individually, whereas a cooperative collection could. For example, consider a genetic algorithm and a best first search. Genetic algorithms excel on problems with fairly smooth search spaces but perform extremely poorly on very "spikey" spaces. Best first search on the other hand has the opposite property. In problems where there is a mix of smooth and spikey surfaces, only a cooperative combination of best first and genetic algorithm could solve the problem (Santos, Shimony, & Williams 1995).

While it seems that there is tremendous promise to cooperative portfolios, it remains very difficult to understand and even possibly analyze the effectiveness of this approach. Even empirical results fail to predict and help address many of the complexities that arise from cooperation. For example, while it seems that the elapsed time for a portfolio algorithm seems to be no worse than the elapsed time for a single best algorithm in the portfolio, communications between algorithms can also worsen performance. In addition to communications latency, communicating "bad" candidate solutions to other algorithms can cause the other algorithms to fall into local optima.

Our *goal* in this paper is to provide the first comprehensive computational model for portfolios of cooperative heterogeneous algorithms. This model is intended to capture all the aspects of how these portfolios behave in order to allow others to conduct rigorous analyses that lead to fundamental results in performance guarantees of effectiveness and efficiency. Most importantly, such results will also provide a better understanding of the intricacies of discrete optimization.

## Computational Model

In order to accurately model the behaviour of a portfolio of cooperative heterogeneous algorithms, we must capture over time the following:

- individual algorithm state changes
  - internal/independent changes
  - changes from interaction
- communications between algorithms
  - message content
  - schedule
- measure of solution quality/performance
  - individual algorithm quality
  - overall portfolio quality

The first item concerns how an algorithm behaves over time without interference as well as how an algorithm incorporates external information (from other algorithms) which also affects state. For example, given a genetic algorithm, it can simply incorporate an external candidate solution directly into its population. For a standard best first search, the simplest incorporation would be as a new pruning value or a more sophisticated change in the search tree.

The key effectiveness of the cooperative portfolio is the actual interactions that occur between algorithms. Thus, the second item addresses the communications schedule and message content which must be accurately captured to reflect the flexibility of many different possible interactions that can potentially occur.

Finally, since our goal is to effectively and efficiently solve discrete optimization problems, a measure over time of the systems performance is naturally required. Such a measure will be fundamental in developing algorithms for determining the optimal interaction schedule.

We begin our formulation as follows:

NOTATION. $\Re$ is the set of real numbers.

Let $P = (\Omega, q)$ be a *discrete optimization problem* where $\Omega$ is the space of candidate solutions for $P$ and $q : \Omega \rightarrow \Re$ is the quality measure we are maximizing. Hence, our goal is to determine $\alpha^* \in \Omega$ such that $q(\alpha^*) \geq q(\alpha)$ for all $\alpha \in \Omega$.

Next, let $D > 0$ be a constant that denotes the overall elapsed time allocated to solving $P$, i.e., $D$ is the time limit set for when an answer must be provided.

We now formally define a cooperative algorithm.

**Definition 1** *A* cooperative algorithm *$A$ for $P$ is a 4-tuple $(\triangle, S, \rho, \nabla)$ where*

- *$\triangle$ is the set of* states *for $A$,*
- *$S : \triangle \times 2^{\Omega} \times \Re^{+} \rightarrow \triangle \times 2^{\Omega}$ is the* state transition *function for $A$,*
- *$\rho : \triangle \times 2^{\Omega} \times (\Omega \cup \{\{\}\}) \rightarrow \triangle \times 2^{\Omega}$ is the* state advancement *function for $A$ and $\rho(\delta, \gamma, \{\}) = (\delta, \gamma)$, and*
- *$\nabla \in \triangle$ is the* start state *for $A$.*

Intuitively, in the state transition $S(\delta, \gamma, \tau) = (\delta', \gamma')$, the algorithm changes from state $\delta$ to $\delta'$ over the execution duration $\tau$. $\gamma$ represents the *working set* of candidate solutions that $A$ is currently considering. $\gamma'$ is the resulting new working set after $\tau$ duration.

$\rho$ is the key to capturing algorithm cooperation. The third parameter for $\rho$ represents the external information from other algorithms that is passed to $A$. $\{\{\}\}$ represents no information being processed by the algorithm.

A cooperative algorithm functions much like a (deterministic) Turing machine. They both require a sequence of inputs that determines their execution. For Turing machines, a sequence of inputs results in a sequence of state transitions and tape content changes. For our cooperative algorithms, our input sequence is an algorithm schedule we define as follows:

**Definition 2** *An* algorithm schedule *$\sigma$ is a finite sequence of ordered pairs from $\Re^{+} \times (\Omega \cup \{\{\}\})$ of the form $\{(\tau_i, \alpha_i)\}_{i=1}^{n}$ such that $0 < \tau_i < \tau_{i+1}$ for $i$ from 1 to $n$. Furthermore, $\sigma$ is said to be* bounded *by $D$ if $\tau_n \leq D$.*

Intuitively, at time $\tau_i$, our cooperative algorithm receives *external* information $\alpha_i$ such as from another cooperative algorithm. Hence, each algorithm in a cooperative portfolio executes according to a schedule. Also, we can easily capture *communications and computation latency* by including $(\tau, \{\})$ in the schedules. These schedules are coordinated into a composite schedule for a portfolio.

Now, for each $\sigma$, we generate an algorithm execution sequence representing the actual execution of our cooperative algorithm under schedule $\sigma$.

**Definition 3** *Given $\sigma$, the* algorithm execution *for $A$ induced by $\sigma$ is denoted by $\bar{A}(\sigma) \subseteq \triangle \times 2^{\Omega} \times \Re^{+}$ such that*

- *$(\delta_0, \gamma_0, \tau_1) \in \bar{A}(\sigma)$ where $\delta_0 = \nabla$ and $\gamma_0 = \{\}$.*
- *For $j$ from 1 to $n-1$, $(\delta_j, \gamma_j, \tau_{j+1}) \in \bar{A}(\sigma)$ where $(\delta'_{j-1}, \gamma'_{j-1}) = S(\delta_{j-1}, \gamma_{j-1}, \tau_j - \tau_{j-1})$. $(\delta_j, \gamma_j) = \rho(\delta'_{j-1}, \gamma'_{j-1}, \alpha_j)$, and $\tau_0 = 0$.*
- *$(\delta_n, \gamma_n, D) \in \bar{A}(\sigma)$ where $(\delta'_{n-1}, \gamma'_{n-1}) = S(\delta_{n-1}, \gamma_{n-1}, \tau_n - \tau_{n-1})$ and $(\delta_n, \gamma_n) = \rho(\delta'_{n-1}, \gamma'_{n-1}, \alpha_n)$.*

Intuitively, from each $(\delta_j, \gamma_j, \tau_{j+1}) \in \bar{A}(\sigma)$ we can determine the parameters for state change computations, $S$, at time $\tau_{j+1}$: $S(\delta_j, \gamma_j, \tau_{j+1} - \tau_j)$. We are simply folding in $\rho$, the state advancement, from the scheduled external input.

We can now define cooperative portfolios and composite schedule as follows:

**Definition 4** *A* cooperative portfolio *$\bar{A}$ is a set of cooperative algorithms $\{A_1, A_2, \ldots, A_m\}$ for $P$.*

Let $\xi = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ be a set of algorithm schedules where $\sigma_i$ is the algorithm schedule for $A_i = (\triangle_i, S_i, \rho_i, \nabla_i)$.

**Definition 5** *$\xi$ is said to be a* composite schedule *for $\bar{A}$ iff for each $(\tau_{i,j}, \alpha_{i,j}) \in \sigma_i$ where $\alpha_{i,j} \neq \{\}$ there exists $(\delta_{k,l}, \gamma_{k,l}, \tau_{k,l+1}) \in \bar{A}_k(\sigma_k)$ where $\tau_{k,l+1} = \tau_{i,j}$, $(\delta_{k,l+1}, \gamma_{k,l+1}) = S_k(\delta_{k,l}, \gamma_{k,l}, \tau_{k,l+1} - \tau_{k,l})$ where $\alpha_{i,j} \in \gamma_{k,l+1}$. Furthermore, $\xi$ is said to be* bounded *by $D$ iff all $\sigma_i$s are bounded by $D$.*

A composite schedule simply guarantees that the input $\alpha_{i,j}$ for $A_i$ originates from some other algorithm $A_k$'s working set at time $\tau_{i,j}$. This now fully captures how algorithms cooperate given composite schedule $\xi$.

## Synchronized Cooperation

Given our computational model above, we can now formally capture the various empirical studies performed by Santos et al mentioned earlier. As an example, we now describe how to model one of the experiments performed.

The experiment involved determining the most probable explanation for a Bayesian network consisting of $h$ random variables $E_1, E_2, \ldots, E_h$. In particular, we are searching for the most probable joint probability over the $E_i$s. Assume that the range of each $E_i$ is denoted by the finite discrete set $R(E_i)$. Hence, our state space is $\Omega = R(E_1) \times R(E_2) \times \ldots \times R(E_h)$.

Six algorithms which consisted of two parameter-variant instances of genetic algorithms, best-first search, simulated annealing, integer linear programming, and an exact barrier algorithm formed the portfolio. Thus, $\bar{A} = \{A_1, A_2, \ldots, A_6\}$ corresponding to each of the algorithms. Algorithms were a mix of minimally interactive (integer linear programming) to more cooperative algorithms such as the genetic algorithms which fully incorporated external solutions. Also, all algorithms were interruptible.

In this example model, we mainly focus on the algorithm communications schedule. One of the problems that Santos et al faced in determining the appropriate schedule was that fact that genetic algorithms were prone to generate many possible candidate solutions quickly where as integer programming took a more methodical approach before a solution is made available. Thus, if the genetic algorithms were permitted to broadcast their solutions as soon as possible, this causes a serious bottleneck in communications and represented a serious interruption to the other algorithms which were forced to process the data.

Instead, Santos et al adopted a synchronized broadcasting schedule for their experiment. Solutions were broadcast by each algorithm (if available) every $\tau$ seconds. Hence, every $\sigma_i$ had the form $\{(j\tau, \alpha_j)\}_{j=1}^{n}$ where $n\tau = D$ and $\alpha_j$ denoted the best solution available from the algorithms. However, if at time $j\tau$, no new solution is found that is better than the one at time $(j-1)\tau$,

$\alpha_j = \{\}$ in order to avoid wasting time. Clearly, the $\sigma_i$s define a composite schedule for this portfolio. Once the composite schedule is executed, we can recover the best solution generated by the portfolio (see below).

## Optimality and Resource Usage

Once our cooperative portfolio $\bar{A}$ completes execution with composite schedule $\xi$, we can recover our final solution as follows: For each $A_i$, define the *solution quality* under $\sigma_i$ to be

$$Q_i(\sigma_i) = \max_{\alpha \in \bar{\alpha}_i} q(\alpha)$$

where $(\bar{\delta}_i, \bar{\gamma}_i) = S_i(\delta_{i,n_i}, \gamma_{i,n_i}, D - \tau_{i,n_i})$. Thus, for composite schedule $\xi$, our *overall solution quality* is

$$Q(\xi) = \max_i Q_i(\sigma_i).$$

**Definition 6** *A composite schedule $\xi^*$ bounded by $D$ is said to be* optimal *if $Q(\xi^*) \geq Q(\xi)$ for all composite schedules $\xi$ bounded by $D$.*

With Definition 6, we have now formally defined the goal of cooperative portfolios to ultimately determine an optimal composite schedule which maximizes the final solution quality. Determining when optimal composite schedules can be found and how they can be constructed will be the key future research emphasis. Our model serves as the basis for formally addressing this problem.

Up to this point, we have implicitly assumed unlimited numbers of processors and unlimited communications bandwidth. Clearly, for our cooperative algorithm portfolios to be practical, we need to consider the resource requirements for implementing our optimal composite schedule. Given our limited space here, we only consider processor usage and guarantees in the following section. We note that communications requirements can be naturally extracted from our composite schedules for analysis.

### Processor Allocation

Processor allocation and usage can be derived for a cooperative portfolio by carefully analyzing the target composite schedule.

**Definition 7** *Given $\xi$, we construct the* schedule graph *$G(\xi) = (V, E, l)$ as follows:*

1. *For each $(\tau_{i,j}, \alpha_{i,j}) \in \sigma_i$ where $1 \leq i \leq n$ and $1 \leq j \leq n_i$, construct $V_{i,j} \in V$.*
2. *Construct $V_{i,0}, V_{i,n_i+1} \in V$ for each $i = 1, \ldots, n$.*
3. *For each $i = 1, \ldots, n$, $j = 0, \ldots, n_i - 1$, add edge $(V_{i,j}, V_{i,j+1}) \in E$ iff $(\delta_{i,j+1}, \gamma_{i,j+1}, \tau_{i,j+2}) \in \bar{A}_i(\sigma_i)$ and either $\delta_{i,j} \neq \nabla_i$ or $\gamma_{i,j} \neq \{\alpha_{i,j+1}\}$. Note $\tau_{i,n_i+1} = D$.*
4. *For each $V_{i,j} \in V$ where $j \leq n_i$ and $\alpha_{i,j} \neq \{\}$, for each $(\delta_{k,l}, \gamma_{k,l}, \tau_{k,l+1}) \in \bar{A}_k(\sigma_k)$ where $\tau_{k,l+1} = \tau_{i,j}$, $(\delta_{k,l+1}, \gamma_{k,l+1}) = S_k(\delta_{k,l}, \gamma_{k,l}, \tau_{k,l+1} - \tau_{k,l})$ and $\alpha_{i,j} \in \gamma_{k,l+1}$, add edge $(V_{k,l+1}, V_{i,j}) \in E$.*
5. *$l(V_{i,0}) = (\nabla, \{\}, 0)$, $l(V_{i,n_i+1}) = (\delta_{i,n_i}, \gamma_{i,n_i}, D)$, and $l(V_{i,j}) = (\delta_{i,j-1}, \gamma_{i,j-1}, \tau_{i,j})$ for $i = 1, \ldots, n$, $j = 1, \ldots, n_i$. $l$ is the* label *for each vertex in $V$.*

Basically, the schedule graph $G(\xi)$ explicitly represents when communications between one algorithm to another takes place. Each communications is represented by a direct edge and each algorithm $A_i$ is represented by a sequence of nodes $V_{i,j}$ which each represent a communications instance at $\tau_{i,j}$ except when $j$ is 0 or $n_i + 1$ which denotes time 0 and $D$ respectively. Also, edges between the nodes for $V_{i,j}$ and $V_{i,j+1}$ represent the results of processing for $A_i$ before communications take place at $\tau_j$. Hence, $G(\xi)$ is a dependency graph.

Given $\xi$, we construct $\xi' = \{\sigma'_1, \sigma'_2, \ldots, \sigma'_n\}$ as follows:

1. Let $T(\xi) = \{t | (t, \alpha) \in \sigma_i$ and $\alpha \neq \{\}\}$. W.l.o.g., let $T(\xi) = \{t_1, \ldots, t_m\}$ where $t_k < t_{k+1}$.
2. For each $\sigma_i$, construct $\sigma'_i$ as follows:
   (a) $\sigma_i \subseteq \sigma'_i$
   (b) For each $t_k \in T(\xi)$ such that there does not exit $(t_k, \alpha) \in \sigma_i$, add $(t_k, \{\})$ to $\sigma'_i$.

**Theorem 1** *$\xi'$ is a composite schedule.*

*Proof.* Follows from our construction above. □

We call $\xi'$ a *normalized* composite schedule for $\xi$. Basically, we are ensuring that if an algorithm receives external input at time $\tau$, then all algorithms also receive input at time $\tau$ via $\{\}$ if necessary. Hence, our schedules are uniform in $\xi'$.

**Proposition 2** $Q(\xi') = Q(\xi)$.

Construct $G(\xi') = (V', E', l')$. For each $t \in (T(\xi) \cup \{0, D\})$, let $v(t) = \{v \in V' | l'(v) = (\delta, \gamma, t)$ for some $\delta$ and $\gamma\}$. Construct $\bar{G}(\xi')$ from $G(\xi')$ by collapsing all the vertices in each $v(t)$ into one node and preserving the edges except for edges between nodes in $v(t)$.

For each composite schedule $\xi$, we define $P(\xi)$ to be the minimum number of processors required to execute $\xi$.

**Corollary 3**

$$P(\xi') = \max_{t \in (T(\xi) \cup \{0, D\})} \textit{in-degree}(v(t)).$$

*Proof.* Follows from Theorem 1 above. □

Since $G(\xi')$ is the dependency graph for our normalized schedule $\xi'$, an edge incoming on a vertex represents a computational dependency requiring processor usage. Thus, the number of edges incident on all vertices $v(t)$ at time $t$ represent the number of processors required at $t$.

**Theorem 4** *$P(\xi')$ is the minimum number of processors needed for $\xi$, i.e., $P(\xi) = P(\xi')$.*

*Proof.* Follows from Proposition 2 and Corollary 3. □

With our processor usage formulated above, we can derive the following theorem:

**Definition 8** *$A$ is said to be* minimally interactive *iff $\rho(\delta, \gamma, \alpha) = (\nabla, \{\alpha\})$ for all $\alpha \neq \{\}$, $\delta \in \triangle$, and $\gamma \in 2^\Omega$.*

**Theorem 5** *If all $A_i$ are interruptible and minimally interactive, then there exists an optimal composite schedule $\xi$ such that $P(\xi) = 1$.*

*Proof.* [Sketch] Let $\xi$ be an optimal schedule and $Q_i(\sigma_i) \geq Q_j(\sigma_j)$ for all $j$. Take node $V_{i,n_i+1}$ and pick any single connected simple path from some $V_{j,k}$ to

$V_{i,n_{i+1}}$ such that there are no incoming arcs to $V_{j,k}$. Construct a new schedule based on this path. □

## Conclusions

We have developed the first formal model for portfolios of cooperative heterogeneous algorithms. Such a model is critical in capturing and analyzing the effectiveness of portfolios for discrete optimization. We can now begin to rigorously address various issues in cooperative portfolios in order to provide performance guarantees critical to future optimization systems. This model will allow us to address the following critical issues:

- What is/should be shared between algorithms?
- How do we use the shared information?
- When do we share the information?
- Which algorithms do we choose to cooperate together?
- When do we change the algorithm portfolio/mix?
- Should we change the mix on-line?

These issues opens up the potential power of cooperative portfolios approach to optimization by exploiting massive parallelism and distributed computing.

The complexity of future problems will no longer permit us to hand custom design algorithms especially in the time-frame they are needed. Instead, cooperative portfolios can provide a natural building blocks approach to, in effect, automatically build customized algorithms based on existing standardized algorithms.

## Acknowledgments

This paper was supported in part by AFOSR Grant Nos. #940006, #9600989, and F49620-99-1-0244 and the University of Connecticut Research Foundation. Special thanks to Eugene S. Santos, Solomon Eyal Shimony, and Edward M. Williams for comments and inspiration.

## References

Baerentzen, L.; Avila, P.; and Talukdar, S. 1997. Learning network design for asynchronous teams. In *Multi-Agent Rationality: Lecture Notes in Artificial Intelligence*, volume 1237. Springer-Verlag. 177–196.

Denzinger, J., and Offerman, T. 1999. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the Congress on Evolutionary Computing*, 2317–2324.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company.

Gomes, C. P., and Selman, B. 1997. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence.*

Hogg, T.; Huberman, B. A.; and Williams, C. P. 1996. Phase transitions and the search problem. *Artificial Intelligence* 81(1-2):1–15.

Hogg, T. 2000. AAAI Workshop on Parallel and Distributed Search for Reasoning Invited Talk.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275:51–54.

Luh, P. B. 1997. Scheduling of flexible manufacturing systems. In Siciliano, B., and Valavanis, K. P., eds., *Control Problems in Robotics and Automation, Lecture Notes in Control and Information Sciences.* Springer-Verlag. 227–243.

Menezes, A.; van Oorschot, P.; and Vanstone, S. 1997. *Handbook of Applied Cryptography.* CRC Press.

Motwani, R., and Raghavan, P. 1995. *Randomized Algorithms.* Cambridge University Press.

Neumaier, A. 1997. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review* 39:407–460.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufmann.

Santos, E. S., and Santos, Jr., E. 1987. Reasoning with uncertainty in a knowledge-based system. In *Proceedings of the International Symposium on Multiple-Valued Logic*, 75–81.

Santos, E. E., and Santos, Jr., E. 2000. Cache diversity in genetic algorithm design. In *Proceedings of the 13th International FLAIRS Conference*, 107–111.

Santos, Jr., E.; Shimony, S. E.; and Williams, E. 1995. On a distributed anytime architecture for probabilistic reasoning. Technical Report AFIT/EN/TR95-02, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Santos, Jr., E.; Shimony, S. E.; and Williams, E. M. 1999. Solving hard computational problems through collections (portfolios) of cooperative heterogeneous algorithms. In *Proceedings of the 12th International FLAIRS Conference*, 356–360.

Santos, Jr., E. 1993. Efficient jumpstarting of hill-climbing search for the most probable explanation. In *Proceedings of International Congress on Computer Systems and Applied Mathematics Workshop on Constraint Processing*, 183–194.

Santos, Jr., E. 1994. A linear constraint satisfaction approach to cost-based abduction. *Artificial Intelligence* 65(1):1–28.

Shimony, S. E. 1994. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence* 68:399–410.

Talukdar, S. N.; Baerentzen, L.; Gove, A.; and deSouza, P. 1998. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics* 4:295–321.

Williams, E.; Santos, Jr., E.; and Shimony, S. E. 1997. Experiments with distributed anytime inferencing: Working with cooperative algorithms. In *Proceedings of the AAAI Workshop on Building Resource-Bounded Reasoning Systems*, 86–91.