Asynchronous Execution and Communication Latency in Distributed Constraint Optimization

Lambert Meertens and Stephen Fitzpatrick*
Kestrel Institute
3260 Hillview Avenue, Palo Alto, California 94304, U.S.A.
meertens@kestrel.edu, fitzpatrick@kestrel.edu

ABSTRACT

Previous papers have reported on a simple, distributed, synchronous algorithm for approximately k-colouring large graphs in soft real time. In this paper, the effects of asynchronous execution and communication latency are investigated. The main conclusions are that strict synchrony is *not* required and that considerable communication latency can be tolerated. These results are important for practical applications of the algorithm involving large networks of low-performance hardware equipped with wireless communication.

1. INTRODUCTION

Large, distributed *constraint networks* arise naturally in many control problems [5]: typically, the variables represent control states of distributed *resources* and the constraints represent physical limitations of the resources (individually or collectively) or requirements imposed by the *tasks* that the resources are expected to accomplish. It is the responsibility of the *control mechanism* to assign values to the variables that (approximately) solve the constraints.

For example, the following application is discussed in detail in [4]:

- In a distributed, wireless sensor network, control variables associated with a single sensor might represent parameters of the sensor's scans such as direction, frequency and intensity.
- Constraints associated with a single sensor might represent limitations on how many directions the sensor can scan simultaneously, or how long is needed to reorient the sensor.
- Constraints associated with multiple sensors might represent problem-specific requirements such as the need for several sensors to scan a given region simultaneously to enhance the overall quality of the data they collect (e.g., when the data is given to a data fusion process).

The lifespans of the tasks impose soft-real-time requirements on the control mechanism. Specifically, a single sensor may have a short scanning range (compared with the size of the network) so that only a small subset of the sensors may be capable of scanning a given target at any particular instant. As the target moves, the subset of sensors within range changes and the control mechanism must adapt the variable assignments to compensate. Adaptation must be quick enough to keep pace with the target; in particular, if the target behaves unexpectedly (e.g., sharply changes velocity) the control mechanism must respond quickly enough to avoid losing the target completely.

A centralized control mechanism is considered to be unrealistic because inter-resource communication has high latency and the number of variables/constraints is too high for the expected computational prowess of the resources. Moreover, a centralized control mechanism would be a single point of failure. Consequently, distributed control mechanisms are of interest.

1.1. ALGORITHM OVERVIEW

Previous papers [2, 3] reported on a simple distributed *synchronous* algorithm for *constraint optimization*. The algorithm has the following characteristics:

- The algorithm is an iterative, anytime process that typically produces approximate solutions: the algorithm immediately generates and publishes a random solution that it iteratively refines, publishing the solution after each iteration. In this way, the control mechanism can quickly initiate an initial response to changing tasks, while optimizing its response as time allows.
- The algorithm may produce a perfect solution (in which every constraint is satisfied) but its primary design objective is to increase the number of satisfied constraints quickly (while they still matter) rather than to satisfy all constraints eventually (by which time the tasks have all changed and the solution is irrelevant).
- The algorithm is fully distributed: each resource determines values for its own variables and communicates them to its neighbours (i.e., nearby resources with

*This work is sponsored in part by DARPA through the 'Autonomous Negotiating Teams' program under contract #F30602-00-C-0014 and the 'Networked Embedded Software Technology' program under contract #F30602-01-C-0123, both monitored by the Air Force Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

which it can interact). The algorithm is thus extremely robust against resource failure.

- The algorithm is scalable: the per-resource, per-iteration costs are proportional to the (mean) degree of the constraint network, not to the size of the network. In typical applications, the mean degree is expected to be bounded because the range over which resources can interact is limited.

1.2. PROBLEM FORMULATION AS GRAPH COLOURING

In [2], the control problem for a distributed sensor network was formulated as a k-colouring problem of a graph's vertices, in which:

- The vertices represent virtual sensors (collections of physical sensors) that are each capable of tracking a target.
- The edges represent mutual exclusion constraints between virtual sensors, that arise from the limitation that a single physical sensor can scan in only one direction at a time.
- The colours represent time slots in a cyclic schedule. Thus, a proper colouring would represent a schedule for the virtual resources in which no two mutually-exclusive sensors were active simultaneously, which translates into a schedule for the physical resources in which no single sensor is expected to scan in two different directions simultaneously.
- The number of colours k corresponds to the length of the cyclic schedule. This is bounded by physical considerations, as follows. A given target must be scanned repeatedly and successive scans cannot be separated by more than a certain amount of time (the maximum *revisit period*) if tracking is to be sustained; thus, the maximum period of a schedule is bounded. But each scan takes a non-zero amount of time (the minimum *dwell time*) so the minimum length of a time slot in a schedule is also bounded. Thus, the number of time slots, and hence the number of colours, is limited.

More generally, a *colour conflict* is an edge that connects two vertices that have the same colour. A conflict represents a glitch in a schedule, since it implies that two virtual resources that are supposed to be mutually exclusive are actually scheduled to be active simultaneously. At the level of the physical resources, a colour conflict represents a single sensor that has been scheduled to scan in two different directions simultaneously. Since the sensor cannot accomplish this, the quality of task accomplishment (target tracking) will be degraded. Thus, the objective of the colouring algorithm/control mechanism may be stated as minimizing the fraction of edges that are conflicts.

This paper carries on the graph colouring formulation of the control problem. However, it should be noted that the algorithm is easily adapted to other types of constraints (simply by redefining what constitutes a conflict) and, in principle, to the optimization of an arbitrary metric over a distributed set of variables; see [4]. Work is ongoing on establishing the algorithm's characteristics for these more general settings.

1.3. ASYNCHRONOUS EXTENSION

In previous papers, execution of the algorithm was strictly synchronous (essentially, it was a systolic algorithm): every resource simultaneously executed a synchronized step in which it updated its own colour; then they all transmitted colour changes to their neighbours. Thus, the communication latency was exactly one.

Given the nature of the hardware anticipated for deployment of the algorithm, requiring strict synchrony is unrealistic (or too costly in communication terms). In this paper, the effect of asynchronous execution is investigated — the concept of a single 'step' of the algorithm is retained in that the mean period between a given vertex's colour updates is fixed. However, the updates are now assumed to be uniformly distributed throughout one period (so the vertices have different offsets or phases determined, e.g., by when their hardware clocks were initialized).

Because the vertices have different phases, the communication latency can have a significant impact on the algorithm's performance. If the latency is short compared with the resources' update period, then the probability of a resource's information about its neighbours' colours being out of date is small — in this case, asynchronous execution may be superior to synchronous execution. However, if the latency is large compared with the update period, then the probability of out-of-date information is high and the algorithm's performance may degrade.

2. SOFT GRAPH COLOURING

Given a set of vertices V, a k-colouring is an assignment of one of k colours (i.e., an integer in Z_k) to each vertex. Let c_v denote the colour of vertex v, where $v \in V$.

Given a set E of undirected edges between pairs of different vertices, a (colour) conflict is an edge $\{u, v\} \in E$ such that $c_u = c_v$. The *degree of conflict* of a colouring is defined as the fraction of edges that are conflicts: $\gamma \equiv |\{\{u, v\} \in E : c_u = c_v\}|/|E|$. A *proper colouring* has no conflicts, so $\gamma = 0$; conversely, $\gamma = 1$ corresponds to a colouring in which every vertex has the same colour.

A *random* k-colouring has an expected degree of conflict of $1/k$. This provides a useful benchmark, particularly for distributed algorithms, since a random colouring can be generated with no communication: it is to be hoped that a non-random colourer can produce k-colourings with $\gamma << 1/k$.

Define the *normalized* degree of conflict by $\Gamma \equiv k\gamma$. Using this metric, a random colouring has an expected value of 1 and a non-random colourer may be gauged to be performing (very) poorly if Γ approaches or exceeds 1.

The experiments reported below are based on $k-$colourable random graphs having specified mean

1. Given a vertex set V, generate a random assignment of the k colours to the vertices.
2. Start with an empty vertex set.
3. Choose at random a pair of vertices that are not yet connected by an edge and that have difference colours; connect these vertices with an edge.
4. Repeat step 3 until the required number of edges have been constructed (where the required number of edges is $|V|d/2$).
5. Erase the colour assignment.

Figure 1: Method for constructing k-colourable random graphs having mean degree d

Each vertex v asynchronously and periodically executes the following process, in which $p \in [0, 1]$ is fixed and is the same for all vertices:

1. Generate a random number $r_v \in [0, 1)$.
2. If and only if $r_v < p$, activate and execute the following:
   (a) Compute a histogram H of colour usage by neighbours.
   (b) Choose any colour c' such that $H(c')$ is minimal.
   (c) If and only if c' is not the current colour, adopt it as the current colour and transmit a message to all neighbours informing them of the change.

Figure 2: The FP algorithm

degree d. These graphs can be constructed using the method shown in Figure 1.

3. THE *FP* ALGORITHM

The algorithm reported in this paper is the *Fixed Probability* (FP) algorithm, in which each vertex repeatedly picks a colour for itself by periodically executing an simple optimization step. In each step, the vertex first randomly determines if it should *activate*. If and only if it activates, the vertex then picks any optimal colour, i.e., one that minimizes its conflicts with the colours it believes its neighbours currently have (based on the messages it has received from them before the start of this iteration). When such a colour choice results in a change of colour (the vertex's current colour may be retained if it is an optimal colour), the vertex communicates the change to its neighbours. Details are shown in Figure 2.

The parameter p is called the *activation probability*. It plays a critical rôle in ensuring that the algorithm converges to a stable colouring (assuming that exogenous characteristics such as the topology remain fixed). If p is high and the communication latency is significant compared with the mean period between steps, then it is likely that a given vertex will activate after a neighbour has changed colour but before the colour change can be communicated. Thus, the vertex will base its own colour choice on partially out-dated information. If this happens frequently enough, the performance of the algorithm degrades. In extreme cases thrashing may result, whereby vertices continually change colour but the quality of the colouring does not improve.

On the other hand, if the activation probability is too low, then the rate at which the colouring can improve is artificially limited (since few vertices change colour in any given step). Thus, it is necessary to balance coherence against parallelism. Previous papers have investigated this balance for synchronous execution of FP. In this paper, the investigation is extended to asynchronous execution including the effect of communication latency.

The FP algorithm is very simple: it may be speculated that more sophisticated versions could, for example, advantageously vary the activation probability over time or from point to point on the network (depending on local characteristics such as the local degree). However, FP has been found to perform surprisingly well over a range of classes of graphs that are important for practical applications. Consequently, it seems worthwhile to investigate its properties before increasing its complexity.

4. THE EXPERIMENTS

In order to simply the experiments, the following assumptions are made:

- Every vertex performs a step of the FP algorithm (which involves stochastic activation, colour choice and initiation of communication) instantaneously and exactly periodically.
- This period does not vary over time and is exactly the same for every vertex. Without loss of generality, this period is taken to be the unit of time.
- Communication latency is precisely determined (by a parameter given to each experimental run): it does not vary over time and is the same for all senders and recipients. The communication latency is measured in units of the FP step period, but it does not have to be an integral multiple of the period.

In applications such as described in the introduction, the step period is expected to vary from vertex to vertex and over time because the anticipated hardware does not provide accurate clocks and, while software can help to improve clock accuracy, the resources are expected to periodically deactivate to conserve energy: this can slightly skew the hardware clocks. However, small amounts of *jitter* are not expected to cause qualitative differences from the results reported here.

On many platforms used for general-purpose computing and control, the communication latency is expected to have considerable variance, caused mainly by variance in the time required for messages to traverse protocol stacks (which variance in turn is caused mainly by multi-tasking in the kernel). However, the hardware anticipated here for deployment of the FP algorithm is so simple that it cannot support a full, multi-tasking operating system. Thus, variance in the latency may well be less than for more complex systems. In any case, a small variance in the latency is not expected to cause qualitative differences from the results reported here.

The experimental methodology is straightforward:

1. Values for the following parameters are chosen: the communication latency (l), the activation probability (p), the number of colours (k), and the mean degree (d).

2. The number of vertices, N, is randomly selected from a range of 900 to 4900. A random k-colourable graph, having N vertices and mean degree d, is constructed.

3. The graph's colouring is randomly initialized. The colouring's normalized degree of conflict is measured/computed. (N.B.: measurement of Γ is performed by the experimental apparatus — it would not be feasible to measure Γ in a practical deployment of the algorithm because it requires gathering information from across the whole network.)

4. The FP algorithm colours the graph for 10000 steps, where a single step involves each vertex being given a chance to activate (and change colour). After each step, the colouring's normalized degree of conflict is measured.

5. Parts 2 to 4 are repeated 20 times for randomly selected graph sizes but for the same latency, activation probability, number of colours and mean degree. The reported results are averages over the 20 graphs/colourings.

The process is repeated for various, deliberately chosen values of l, p, k and d.

4.1. RESULTS

Experiments with synchronous execution of the FP algorithm show that an activation probability of around 0.3 is typically a good value for a wide range of graphs: it achieves a quick reduction in the number of conflicts and is stable in the long term. Consequently, the presentation of the experimental results for asynchronous execution begins with the results for $p = 0.3$,

Figure 3 shows the normalized degree of conflict against the number of steps taken (averaged over 20 colourings) for $p = 0.3, d = 10$ and $k = 3$ and for various latencies. Results for synchronous execution are included for comparison. Several observations may be made:

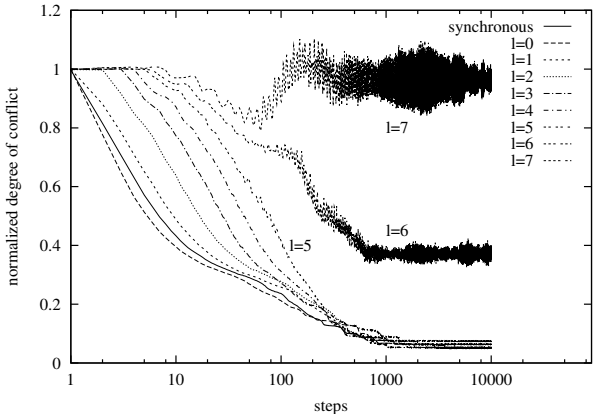

Figure 3: Γ vs. steps for $d = 10, k = 3, p = 0.3$

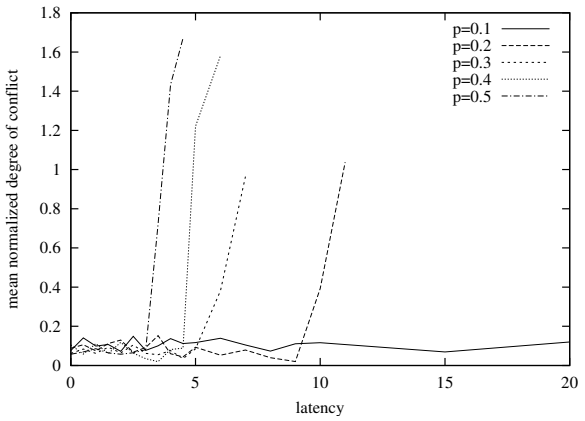

Figure 4: $\bar{\Gamma}$ vs. l for $d = 10, k = 3$

- As the latency increases from zero, the short-term performance degrades (i.e., the degree of conflict is higher for the first few dozen steps). This is expected: increasing the communication latency should slow down the algorithm.

- However, the long term performance is hardly affected by communication latencies up to 5 (where the latency is measured in units of the period of the FP algorithm).

- For higher latencies, the long term performance sharply becomes very poor; for a latency of 7, the performance is about the same as for random colouring.

In order to assess the effect of latency further, the mean of the normalized degree of conflict can be computed over the steps of the FP algorithm: $\bar{\Gamma} \equiv \sum_{i=1}^{S} \Gamma_i / S$, where the number of steps S is always 10000 in the experiments reported here, and Γ_i is the normalized degree of conflict after step i.

Figure 4 shows the mean normalized degree of conflict against latency for various activation probabilities. Clearly, for each activation probability, there is a threshold latency above which the performance dramatically degrades. Moreover, this threshold varies inversely with the activation probability. This is not surprising since the average number

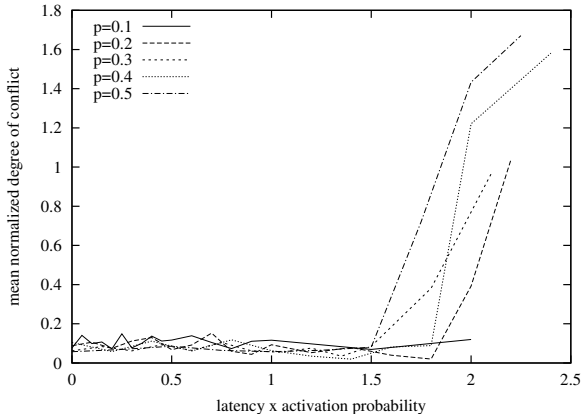

Figure 5: $\bar{\Gamma}$ vs. lp for $d = 10, k = 3$

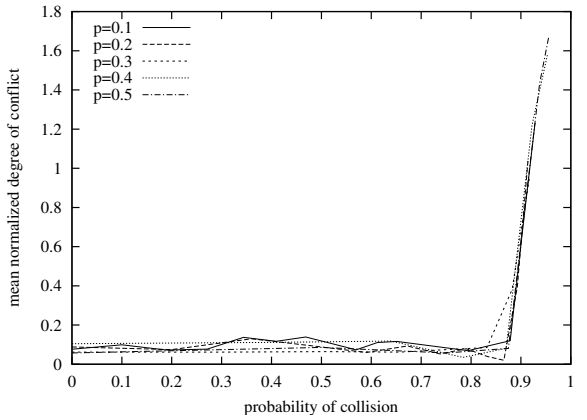

Figure 6: $\bar{\Gamma}$ vs. p_{collide} for $d = 10, k = 3$

of times a vertex is expected to activate during some time period is proportional to the activation probability — the higher the probability, the more frequently a vertex activates during a period equal in length to the communication latency, and the more likely a vertex is to have out-of-date information when updating its colour.

This reasoning suggests scaling the horizontal axis by the activation probability, as shown in Figure 5. This figure shows that at the point where the mean normalized degree of conflict becomes 1, lp is approximately constant (i.e., the latency and activation probability are approximately inversely proportional). This reinforces the hypothesis that the main factor in the degradation of the performance is how many times a vertex activates on average during a period of length one communication latency.

However, the figure also shows a trend favouring lower activation probabilities: when p is low, the algorithm can tolerate latencies higher than would be expected from the above considerations. A simple probabilistic argument gives the probability p_{collide} that when a vertex changes colour, at least one of its neighbours has changed colour within the preceding l time units as $p_{\text{collide}} \approx 1 - (1 - p)^l$.

Figure 6 shows the mean degree of conflict plotted against p_{collide} for various latencies and activation probabilities.

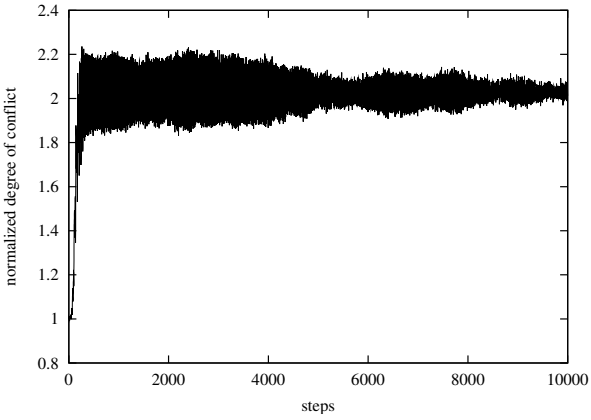

Figure 7: Γ vs. steps, $d = 10, k = 3, p = 0.3, l = 15$

d	k	p	l	$\bar{\Gamma}$
10	3	0.3	15	2.01
10	3	0.3	30	2.17
10	3	0.3	50	2.18
20	7	0.1	40	1.85
30	12	0.2	80	1.96

Figure 8: $\bar{\Gamma}$ for high-latency

The plots are clearly approximately consistent, but insufficient data has been collected to assess the consistency quantitatively. Moreover, while qualitatively similar results are found for other classes of graph (e.g., having higher d and k), the threshold at which the mean normalized degree of conflict sharply increases is not the same for each class of graph. Further experiments would need to be performed to determine: (1) the relationship, if any, between the threshold and, say, d and k; (2) whether the threshold corresponds to a sharp phase transition or a smooth, albeit steep, increase in conflicts.

4.2. VERY HIGH LATENCIES

When the latency is very high ($lp >> 1$), the FP algorithm behaves rather surprisingly: in particular, the normalized degree of conflict tends to a mean value of approximately 2 (although with strong fluctuations), regardless of the number of colours, mean degree and activation probability. For example, Figure 7 shows the normalized degree of conflict against steps for a latency of 50 (with $d = 10, k = 3, p = 0.3$) and Figure 8 shows values of the mean normalized degree of conflict for various parameters (all averaged over 20 colourings).

Figure 9 shows even more surprising results. It shows what fraction of vertices have each colour after every step of the algorithm: the usage of each colour rises and falls cyclically over time, with each becoming dominant in turn (note that at the peaks of the cycles, almost every vertex has the same colour). Similar results are obtained for higher numbers of colours, although the peaks can be less regular/well formed.

Analysis of these results suggests that the time lag corre-

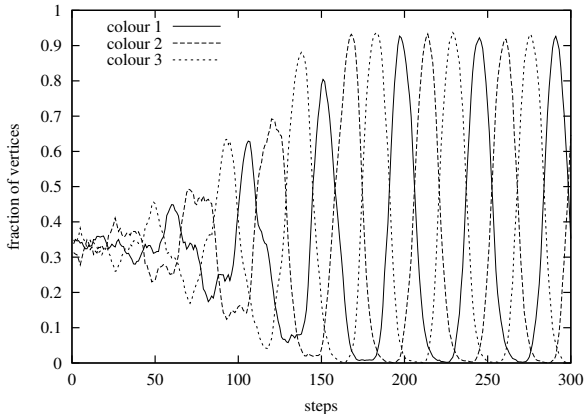

Figure 9: Fraction of vertices having each colour ($d = 10, k = 3, p = 0.3, l = 10$)

sponding to the communication latency causes the control mechanism to be caught in an out-of-phase, oscillating trajectory. The period of oscillation appears to be at least twice the latency, corresponding to a phase lag of less that π.

These results are interesting and warrant further investigation, but their immediate practical ramification is that good performance under high-latency communication requires low activation probability; i.e., the probability of collision needs to be kept below a critical threshold, as discussed above.

5. CONCLUSIONS

Two qualitative results are strongly supported by the experiments reported here: (1) the FP algorithm can be effectively executed asynchronously, and (2) it can be readily adjusted to accommodate communication latency through a combination of increasing its period and reducing its activation probability.

The most important quantitative result relates to the threshold in the dependence of the mean normalized degree of conflict on the probability of collision — it is anticipated that plots such as Figure 6 can be compiled for combinations of problem-specific characteristics such as the mean degree of the constraint network and the number of colours, and the plots used to relate the communication latency to how frequently vertices should change colour.

For example, in a practical application the communication latency may be fixed by the choice of hardware and the step period of the FP algorithm may be fixed by other periodic tasks that must be serviced. Plots such as Figure 6 can then be used to determine optimal values for the activation probability. The precise definition of optimality may be problem-specific, but two typical cases are:

- If short-term reduction in the number of conflicts is of overwhelming importance, then the activation probability should be set so that the probability of collision falls just below the threshold. This will guarantee responsiveness in the control mechanism.
- If low communication cost is more important than short-term reduction in the number of conflicts, then the activation probability should be set to a moderate value such that the probability of collision is well below the threshold.

Many more experiments will need to be performed before such determinations are feasible in general. Nevertheless, the qualitative results are important because they greatly simplify the deployment of the FP algorithm on cheap, low-performance hardware; if the reported results did not hold, a method for tightly synchronizing the hardware units would need to be implemented.

6. RELATED WORK

The Fixed Probability algorithm is an extension of a deterministic algorithm described in [1]. Yokoo [5] describes a distributed constraint satisfaction algorithm called Distributed Breakout (DB) that is based on localized manipulation of edge weights, intended to avoid stagnation in local optima; experimental comparison of the FP and DB algorithms is reported in [6].

REFERENCES

[1] Marko Fabiunke. Parallel Distributed Constraint Satisfaction. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, volume III, pages 1585–1591. CSREA Press, 1999.

[2] Stephen Fitzpatrick and Lambert Meertens. An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs. In Kathleen Steinhöfel, editor, *Symposium on Stochastic Algorithms: Foundations and Applications, Proceedings of SAGA 2001*, number 2264 in Lecture Notes in Computer Science, pages 49–64. Springer-Verlag, 2001. Berlin, Germany.

[3] Stephen Fitzpatrick and Lambert Meertens. Scalable, Anytime Constraint Optimization through Iterated, Peer-to-Peer Interaction in Sparsely-Connected Networks. In *The Sixth Biennial World Conference on Integrated Design & Process Technology*, June 2002. Pasadena, California, U.S.A. (to appear).

[4] Lambert Meertens and Stephen Fitzpatrick. Peer-to-Peer Coordination of Autonomous Sensors in High-Latency Networks using Distributed Scheduling and Data Fusion. Technical Report KES.U.01.09, Kestrel Institute, Palo Alto, California, December 2001.

[5] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Trans. on Knowledge and Data Engineering*, 10(5):673–685, September/October 1998.

[6] W. Zhang, G. Wang, and L. Wittenburg. Distributed Stochastic Search for Implicit Distributed Coordination: Parallelism, Phase Transitions and Performance. In *AAAI-02 Workshop on Probabilistic Approaches in Search*, 2002.