Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002
©2002 Society for Desing and Process Science

Scalable, Anytime Constraint Optimization through Iterated, Peer-to-Peer Interaction in Sparsely-Connected Networks

Stephen Fitzpatrick and Lambert Meertens
Kestrel Institute
3260 Hillview Avenue, Palo Alto, CA 94304, USA
fitzpatrick@kestrel.edu & meertens@kestrel.edu

ABSTRACT: This paper reports on an algorithm for anytime, stochastic, distributed constraint optimization that uses iterated, peer-to-peer interaction to try to achieve rapid, approximate solutions to large constraint problems in which the constraint variables are naturally distributed. Two examples are given — graph coloring and coordination of distributed sensors — together with experimental results on performance.

I. Introduction

Dynamic, distributed constraint optimization problems arise naturally in high-latency networks of loosely-coupled nodes that must collaborate to accomplish some time-varying set of tasks. The objective is to determine a time-dependent mapping from a set of variables (representing some control states of the nodes) to allowable values, such that the mapping optimizes some trade-off between the quality of task accomplishment and the costs of the nodes' actions (which result from setting the variables to the specified values). The allowable values are determined by constraints on an individual node's variables and between variables on separate nodes.

The variables are naturally distributed because they are tightly coupled to physical nodes that are themselves distributed, and the communication latency is too high to permit effective remote control.

When the set of tasks to be accomplished is dynamic (e.g., individual tasks change or the number of tasks changes exogenously), a network may be required to respond in soft real time.

For example, small, cheap, autonomous, battery-powered sensors and actuators equipped with low-power radio transmitters/receivers enable the deployment of large, scalable sensor/effector networks, with the following typical characteristics:

- Each node in the network accomplishes some local, simple task, such as acquiring a single measurement of a nearby, moving target.
- A single node's actions may be rather constrained. For example, a single sensor may be able to take measurements in only one direction at a time.
- A single node may be unreliable, but it is likely that a nearby node can substitute when a given node fails.
- Multiple, nearby nodes need to collaborate to accomplish *mission-level tasks* such as tracking a moving target over an extended period.
- The number of nodes may be high. For example, networks containing 10^5 sensors are currently envisioned [7].
- Communication is limited: bandwidth is low (compared with wired networks), range is limited (compared with the size of the network) and latency is high (in part because the underlying communication mechanism has high latency, but also because many nodes may contend for a limited number of communication frequencies).
- Energy conservation is important. A sensor network may be required to operate unattended for an extended period, and the energy costs associated with actions such as active sensing (e.g., emitting a radar beam) or transmitting messages are significant.
- Exogenous parameters can be predicted with reasonable reliability, but soft-real-time responsiveness is required when the parameters change in unexpected ways. For example, a target's trajectory may be projected, say, fifteen seconds into the future with reasonable certainty, but occasionally the target may sharply change velocity and then the network may be required to adapt its actions to the new trajectory in a few seconds.

Such networks are expected to be cheaper to deploy and maintain than traditional sensor systems (comprised of a few, high-performance nodes). However, the problem arises of how to achieve good, scalable, autonomous, soft-real-time resource coordination: the value associated with mission-level tasks (such as tracking a target) is high so the quality of task accomplishment is important, but the individual sensors are limited in their sensing capabilities (requiring multi-sensor collaboration) and energy is limited (implying that collaboration must be 'intelligent' — flooding each target with sensor energy would be wasteful).

Centralized coordination is expected to be unscalable, because resource management is typically combinatorially hard (so solving a problem for 10^5 nodes in soft-real-time

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.

would require considerable computing prowess), a centralized coordinator would require communication with all parts of the network (resulting in high latency due to the range of a single transmission, even if the communication bandwidth is sufficient), and a centralized controller would represent a single point of failure (the controller's failure would disable the entire network).

Strict decomposition of the network into fully-independent regions, each small enough that communication latency and combinatorial complexity are manageable, is typically not possible because, for any particular decomposition, it is typically possible to find mission-level tasks whose optimal accomplishment requires collaboration between sensors in the supposedly independent regions. (For example, if the size of the independent regions is determined by communication latency, then target trajectories would rarely be contained within a single region.)

Of course, decomposition may be approximate: for example, collaboration may occur between regions as well as within individual regions. The issue to be addressed then would be how to coordinate the regions while maintaining fault tolerance but without drastically increasing communication. For example, if each region has a designated leader responsible for coordination with other regions, then sensor coordination involves several levels of communication (sensor-leader and leader-leader) and failure of the leader results in failure of the whole region in naïve implementations.

While region-based coordination cannot be dismissed as readily as centralized coordination, it seems natural to pursue a coordination topology that mirrors the communication and collaboration topologies. Because a sensor's measurements and communication have limited range (extending over meters rather than kilometers), a given sensor typically collaborates with other sensors that are geographically close to it. This suggests that a sensor should coordinate directly with nearby sensors, in a fully-distributed, peer-to-peer fashion. While there are similarities with the regional decomposition approach, the main difference is that there are no artificially imposed boundaries.

Peer-to-peer coordination is expected to be fully scalable (the communication and computational costs of a single node are determined by the number of sensors within interaction range, not by the total number of sensors in the network) and extremely fault tolerant (the failure of a single node should result in little degradation in performance of the overall network). Moreover, because long-range, high-latency communication is avoided, it *may* be possible to achieve soft-real-time responsiveness.

The central problem, and the main topic of this paper, is to ensure high-quality accomplishment of mission-level tasks using peer-to-peer coordination.

II. A Peer-to-Peer Constraint Optimization Algorithm

The primary characteristic of the distributed constraint optimization algorithm described in this paper is that *each node makes its own decisions*: i.e., each node determines the values of its own variables. For example, in a sensor network, each sensor node determines what measurements it will take.

Of course, in order to make good decisions, a node needs to know what is occurring in the real world and what are the near-term intentions of those nodes with which it interacts. ('Near-term' means far enough into the future that communication latency is relatively insignificant.) For example:

- A sensor node may operate in different measurement modes (target acquisition or tracking) depending on whether or not a target is nearby; more precisely, the measurement mode may be determined by the node's *knowledge* of target trajectories.
- If a sensor node knows that only one other nearby sensor intends to take a measurement of some specific target, it may decide to also take a measurement of that target, and to do so at the same time as the other node, to enhance the quality of their combined measurements.
- Conversely, if a sensor node knows that two other nodes will both be taking measurements of some target, then the node may decide to turn off its sensors to conserve energy.

Thus, the design paradigm for peer-to-peer optimization is to determine what information a node needs to make good decisions by itself, and to determine how to provide that information to each node with sufficiently low latency and in a cost-effective manner. (Of course, that will not be possible for some problems.)

For example, in a sensor network, each node may periodically broadcast the measurements that it has acquired, so that each node has sufficient information about the real world in its vicinity to compute estimates of trajectories of nearby targets. Each node may also periodically broadcast its intentions regarding what measurements it will take in the near future, so that each node can adjust its own actions to best match/complement those of its neighbors.[1]

Having such information at hand, each node adjusts its own variables to optimize a local version of the global problem quality metric. Since a single node may not be able to ensure that all constraints involving its variables are satisfied, account must be taken of violated constraints. These can be incorporated into the quality metric by assessing a penalty for each violated constraint, which penalty may depend upon the degree of violation. (In other words, *soft constraints* are used.)

For example, if the global metric is based on tracking all targets well and reducing energy expenditure, then each

[1] Broadcasting is assumed to be restricted by the physical range of low-power radio transmissions, so a broadcast message reaches only a small number of nearby nodes. Broadcasting is thus scalable.

node adjusts its own variables so that the targets it knows about are tracked well and its own energy expenditure is minimized.

Decision making (i.e., variable assignment) and information dissemination are ongoing tasks: if exogenous parameters remain relatively stable (e.g., targets move in straight lines) then the quality of the global solution is expected to improve over time as nodes repeatedly perform local optimizations; if exogenous parameters change (e.g., a target changes direction) then adaptation occurs first where the change is directly measured and then ripples through the network (as nodes disseminate new information about the real world and new values for their variables).

Moreover, each node always has available *some* values for its local variables, and so it is always able, when helpful, to participate in accomplishing mission-level tasks. In other words, the physical actions of the network (e.g., taking measurements) are *not* postponed for some indefinite period, waiting for network coordination to determine an ideal solution to a large combinatorial problem.

A. Issues

There are, of course, several potential drawbacks associated with the approach described above.

A.1 Problem does not fit

Peer-to-peer optimization will not be suitable for all network problems. For example, a later section shows how to use the algorithm for coloring sparse graphs, but the algorithm would not be suitable for coloring dense graphs: for example, if used to color a complete graph, each node would acquire a local problem that is identical to the original global problem.

A.2 Coherence/convergence

Each node is continually adjusting its own behavior based on its beliefs regarding other nodes' intended behavior. Since those other nodes are also adjusting their behavior, it is possible for *incoherence* to result when the rate of change approaches the rate at which information about changes can be disseminated.

In extreme cases, the quality of the global solution may be worse than, say, a random solution. Moreover, the nodes may not converge to a stable solution, and may continually expend energy communicating new values without improving the solution — this situation may be described as *thrashing*.

In the algorithm reported here, a simple stochastic technique is used to reduce the number of nodes that may change their values at any given time and thus help achieve convergence. Of course, if the reduction were too severe, the adaptivity of the network would suffer: coherence and adaptivity must be balanced.

A.3 Global optimality not assured

Even if all of the nodes converge to a stable set of values for their variables, in which each node's values are optimal given the other nodes' values, the global solution may be sub-optimal. This defect may be unavoidable if only local interaction is allowed and real-time adaptivity required. The practical questions are whether the solutions are good enough in actual use, and whether another algorithm can produce better solutions in the time available.

B. Algorithm Details

Both the sensor example and the graph coloring example described below can be formulated abstractly as follows:

- In a network of nodes N,
- each node u must determine a value for a local variable x_u (which may be a complex data structure such as a schedule of measurements),
- such that x_u optimizes a locally-computable metric $\mu(x_u, \{x_v^u\})$, where x_v^u represents u's knowledge of v's variable. The metric may include terms for task accomplishment, penalties for violated constraints, and costs of operating the network.

The proposed algorithm, *FP(p)* where $p \in (0, 1]$ is the fixed *activation probability*, is an iterative, hill-climbing algorithm that operates as follows:

- Each node u (quasi-)periodically determines if it should *activate* by generating a random number $r_u \in [0, 1)$ and comparing with p: the node activates iff $r_u < p$. (For the graph coloring example, a 'conservative' constraint is added: a node will not activate if it already has an optimal solution. It is not yet clear for which other problems this conservative constraint will be useful.)
- If the node activates, it determines a value for its local variable x_u that is optimal according to its metric μ, given what it currently knows of the variables of nearby nodes. For simple data structures, it may be possible to identify all optimal values, in which case one is selected at random. For complex data structures, optimality may be local in the hill-climbing sense (the current value of x_u gives a higher metric value that any of the values adjacent to x_u, but there may be values further away that would give better metric values).
- If the node activates and changes its variable, it broadcasts the new value to nearby nodes.

The initial values of the variables are determined simply, by randomization for example.

The following sections show how to apply this algorithm to two examples: distributed sensor coordination and distributed graph coloring.

III. Distributed Sensor Coordination Example

The distributed sensor coordination example shows how the peer-to-peer optimization algorithm can be applied to a more-or-less realistic application. The experiments that have been performed to date for this example are proof-of-concept experiments. In-depth experiments into the dynamics of the algorithm are discussed below in the distributed graph coloring example.

The objective of the resource manager (i.e., the distributed constraint optimization algorithm) in this sensor network example is to optimally coordinate the actions of sensors so as to obtain measurements that are input to a target tracker to produce a *world estimate* — i.e., a probability distribution/density function over the states of possibly multiple targets (e.g., positions and velocities).

Conceptually, an appropriate metric for gauging the performance of the resource manager would be some measure of how accurately the target states are known (e.g., the 'width' of the probability density function) compared with the costs of taking the measurements.

However, determining how a particular measurement will affect an estimate can be rather complex, so a *proximate* metric may be substituted which attempts to directly characterize the *measurement quality*. For example, typically the quality of a measurement is expected to drop as the distance between a sensor and a target increases.

Thus, given a current world estimate and a proposed set of measurements, the (global, conceptual) quality metric:
1. predicts where targets are expected to be when the measurements are taken by *evolving* the world estimate according to some *model of evolution*;
2. determines the quality of each measurement based on the expected target positions;
3. combines the single-measurement qualities into an overall quality for each target;
4. combines the single-target metrics into a single, overall metric for the entire proposed set of measurement.

The resource manager attempts to determine a set of measurements that optimizes the trade-off between the measurements' quality and cost [6].

In practice, a world estimate may be considerably simplified to reduce computational costs, and for simplicity of this discourse, it is assumed that target states are precisely known and their evolution is deterministic. Thus, a world estimate W_t at time t is represented as a pair of maps from target index $i \in I$ to exact target positions P_t and velocities V_t: $W_t \equiv \langle P_t, V_t \rangle$.

Under this formulation, the above steps become:
1. Given a proposed set of measurements and times for performing the measurements, $M \equiv \{\langle m_j, t_j \rangle\}$, the expected position of each target is computed at each time: $\vec{p}(i, t_j) = \vec{P}_t(i) + \vec{V}_t(i).(t_j - t)$.
2. The expected quality of each measurement is computed based on the target positions. Since each measurement can acquire information about any subset of the targets, the quality metric for a single target is expressed as a map from target to, say, real numbers: $q(W_t, m_j, t_j) = \{i \in I \rightarrow q_0(\vec{p}(i, t_j), m_j)\}$ where q_0 is some function capturing the physical realities of sensor-target measurement quality.
3. Individual measurement qualities are then combined component-wise:

$$Q(W_t, M) = \{\, i \in I \rightarrow Q_c(\{q(W_t, m_j, t_j)(i) | \langle m_j, t_j \rangle \in M\})\} \ .$$

The combination function Q_c can have arbitrary form, but the intention is that it should reflect the value of sensors collaborating on measuring a single target. For example, it may award a low value if only two far-off sensors measure a target, a high value if two or three near sensors measure a target, but only a slightly higher value if more than three sensors measure a target — see Figure 1. In this way, lack of collaboration is penalized, as is swamping of a single target by many sensors (since the increase in quality is low compared with the increase in costs).
4. The overall metric represents some cumulative value, e.g., the sum, of the single target metrics: $\mu(W_t, M) = \sum_{i \in I} Q(W_t, M)(i)$.

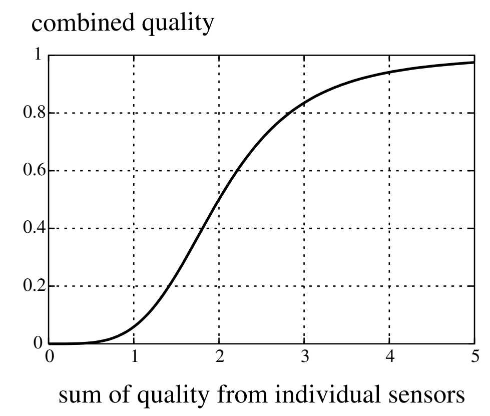

Fig. 1. Non-linear function for combining measurement qualities: $y = x^4/(2^4 + x^4)$

In practice, the measurement quality metrics may account for multiple-target interference (e.g., one target obscuring a sensor's view of another target) and lack of simultaneity of measurements: both of these effects are easily accommodated in this formulation.

Terms should also be included to account for violated constraints and energy expenditures: these are straightforward and are not discussed further.

A. Localizing the Metric

Having defined the global metric for resource management, a local, single-node metric can be formulated to allow a node to optimize its own behavior. The local metric is essentially the same as the global metric, but is restricted to the information that a single node has at hand, namely:
• Each node has an estimate of the states of *nearby* targets. In order for each node to be able to compute its estimate,

each node periodically broadcasts its measurements (which are received by all nodes within transmission range). Each node combines its own measurements with those it receives from nearby nodes to maintain its world estimate.

• Each node knows, with some latency, what measurements nearby nodes intend to take in the near future; i.e., it has nearby nodes' *schedules*. As with the measurements, this is achieved by each node periodically broadcasting its own schedule.

Thus, the objective of each node is to compute a schedule of measurements that it will take in the near future, such that the schedule's combination with the schedules of nearby nodes is optimal with respect to the node's current world estimate.

Let W_t^u be node u's world estimate, and M_v^u be what u believes is v's measurement schedule. Then node u is to compute its own schedule M_u of measurements such that $Q(W_t^u, M^u)$ is optimal, where $M^u \equiv M_u \cup \bigcup_v M_v^u$ is the combined schedule known to u. The details of how each node optimizes its own schedule are not considered here, but some variant of hill-climbing seems suitable.

B. Experiments with a Simulated Sensor Network

The experiments carried out for this example, to date, are small-scale, proof-of-concept experiments, based on a simulator of a network of simple radar sensors and one or two moving targets. Each sensor has three radar beams but only one analogue-to-digital converter: the beams can be activated in any order, but only one of them can be sampled at any given time. A beam uses energy while it is active (even if it is not being sampled), and there is a significant latency between a beam being activated it stabilizing sufficiently to give reliable measurements. A resource manager such as described above is used to determine which measurements should be taken and when beams should be activated and deactivated.

In these small-scale experiments, each node computes nearly identical world estimates. The target positions estimated by one of the nodes is compared with the true target positions (as recorded by the simulator).

Figure 2 shows results for 8 sensors and 1 target; Figure 3 shows results for 8 sensors and 2 targets. The (simulated) room size is 40×40 meters^2. The triangles represent the positions and orientations of the sensors. The black circles represent position estimates produced by the tracker. The gray lines obscured by the black circles represent the true target paths.

The error in a position estimate $\vec{p}$ at time t can be computed as the distance between $\vec{p}$ and the true target position $\vec{g}$ at time t. The track quality can be measured as the root-mean-square of the single-estimate errors. For a single target, r.m.s. errors of about 0.6 meters are obtained; for two targets, about 0.9 meters. These values are close to the limit (about 0.5 meters) of what the tracker is expected to be able to achieve, given its approximation techniques, its model of the targets, and (simulated) sensor noise.

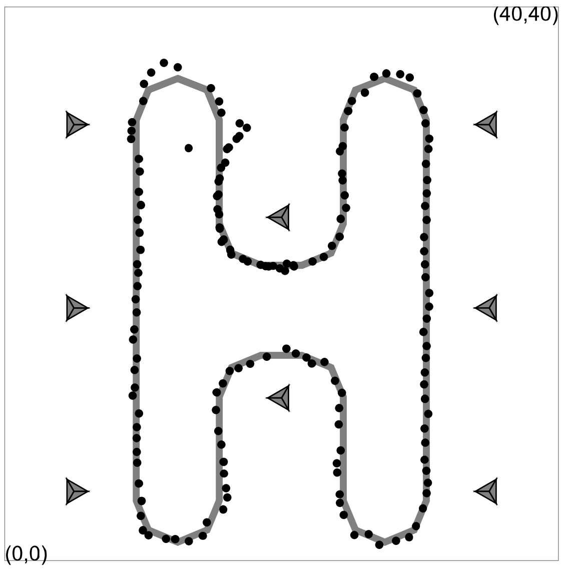

Fig. 2. Tracking a single target

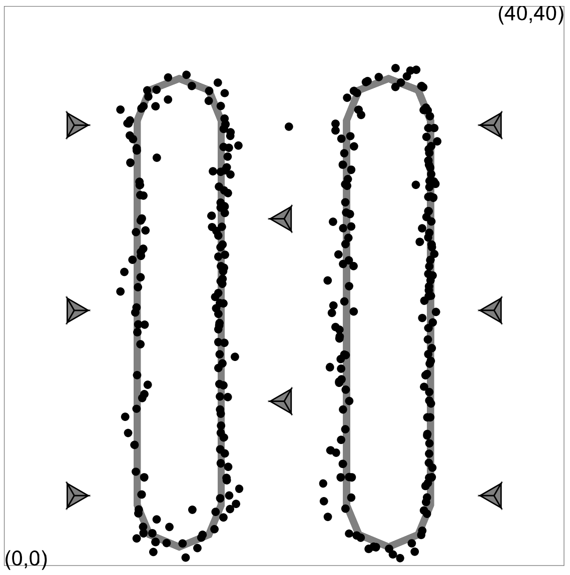

Fig. 3. Tracking two targets

The mean power usage is about 50% (meaning that, on average, half the beams are on at any given time). This is slightly higher than would be hoped for, especially for a single target, for which it should be possible to reduce the power usage to around 33% (meaning that only one beam is active on each node, on average).

IV. Algorithm Dynamics: Experiments with Distributed Graph Coloring

The sensor coordination example shows how a real-life problem can be solved using peer-to-peer constraint optimization. However, it is not straightforward to isolate the performance of the constraint optimization algorithm from the performance of the tracking algorithm. To better investigate the former, experiments can be performed with distributed graph coloring.

Given an undirected graph G with node/vertex set N and edge/arrow set E, a k-coloring C_k is an assignment of one of k 'colors' to each node: $C_k \equiv \{v \in N \to c_v \in Z_k\}$. The quality metric on an coloring is the *degree of conflict*, defined as the fraction of edges that are conflicts (i.e., that connect nodes of the same color): $\gamma \equiv |\{\{u,v\} \in E | c_u = c_v\}|/|E|$.

This metric has a range from 0 (no conflicts, corresponding to a *proper* coloring) to 1 (all conflicts, corresponding to a single-color coloring). A randomly generated coloring has an expected metric of $1/k$: this provides a useful baseline for non-random colorers, since a random coloring can be computed without inter-node coordination.

The degree of conflict is a global metric. In the peer-to-peer colorer described here, each node chooses its own color so as to optimize a local metric. In order to define a suitable local metric, the following two assumptions are made:

- Each node knows who its neighbors are. That is, each node u knows $E^u \equiv \{v \in N | \{u,v\} \in E\}$.
- Aside from its own color, each node also knows, with some latency, the color chosen by each of its neighbors. Let c_v^u ($v \in E^u$) denote the color that node u believes node v has.

Then the local degree of conflict is defined as $\gamma^u \equiv |\{\{u,v\} \in E^u | c_u = c_v^u\}|/|E^u|$. The peer-to-peer colorer is based on each node minimizing γ^u: i.e., when a node chooses a color for itself, it chooses any color that it believes is least used among its neighbors.

Simple experiments can be performed to investigate the dynamical properties of the distributed constraint optimization algorithm — scalability, stability, short-term conflict reduction and long-term convergence. In the experiments reported below, the colorer is synchronous: each node executes an activation/color-selection step simultaneously, then executes a communication step (if needed) simultaneously.

Preliminary experiments have also been performed with an asynchronous colorer — most of the results carry across, but convergence is simpler to achieve in the asynchronous case provided that the communication latency is significantly lower than the mean period between the nodes' activation steps. (In effect, asynchronous execution results in fewer nodes changing color simultaneously, resulting in better coherence.)

A. Overview of the Experiments

The graph coloring example provides simple, clean metrics for investigating various aspects of the performance of the FP algorithm, namely:

- **Long-term convergence** — the algorithm is allowed to run for a relatively long time on a fixed graph and the degree of conflict measured at the end of the run. This value indicates how well the algorithm performs under stable conditions; for example, it might be hoped that it would eventually achieve a proper coloring, if the number of colors is at least the chromatic number.
- **Short-term conflict reduction** — the degree of conflict is measured over a relatively short time as the algorithm runs on a fixed graph. This experiment is probably the most critical, since it reflects the real-time responsiveness of the algorithm.
- **Scalability** — the degree of conflict is measured as the algorithm runs on graphs of various sizes. This allows the effect of graph size to be quantified.
- **Robustness** — as the algorithm runs, the degree of conflict is measured and the graph's topology is varied in such as way as to (likely) maintain the chromatic number. The topology changes can be minor and continual (showing how the algorithm responds to fluctuating unreliability in a system) or major and intermittent (showing how the algorithm responds to wide-scale failures).

These performance aspects were investigated in several experiments, as described below.

B. Effect of Activation Probability

The performance of the FP(p) algorithm depends critically on the activation probability p. For example, Figure 4 shows the degree of conflict over time for various values of p. The results shown are averages for 2-dimensional Cartesian networks; i.e., graphs with nodes at (i,j) where $i,j = 0,1,2,\ldots$, and with edges between neighboring nodes along either axis or diagonal.[2] If edge effects are ignored, such graphs have mean degree 8 and chromatic number 4 — 4 colors were used for the experiment. Similar results were obtained for other, regular graphs and for random graphs of higher chromatic number.

Note that the degree of conflict has been normalized by multiplying it by the number of colors; this produces a scale on which a random coloring has an expected value of 1.

For high p, the algorithm converges slowly — for very high p (on complex graphs) the algorithm may produce colorings that are worse than random, and may not converge. For lower p, the algorithm converges to a low value, but does not reach zero in the time allowed, even though the

[2]Similar results have been obtained in experiments on sparse, random graphs of higher chromatic number.

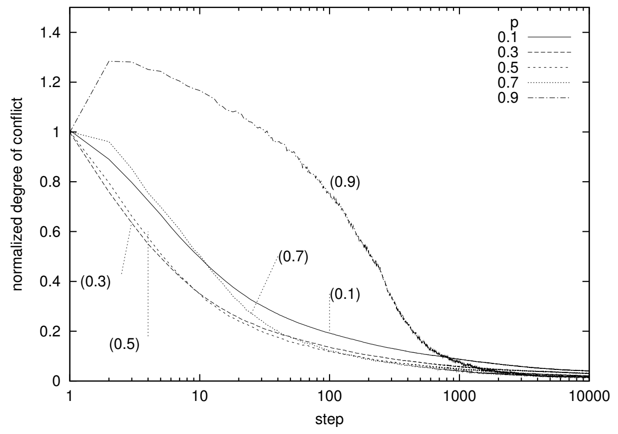

Fig. 4. Effect of activation probability

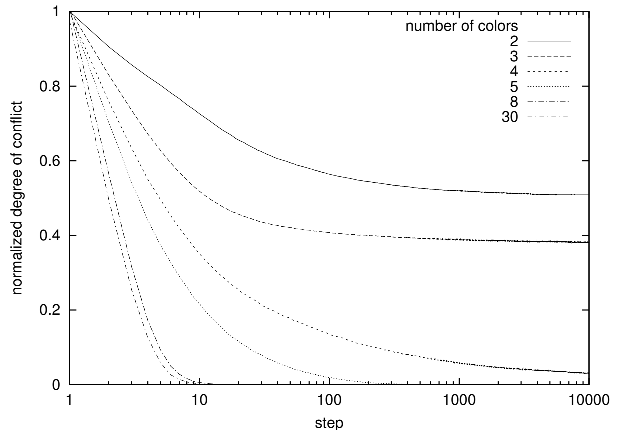

Fig. 5. Effect of number of colors

number of colors is equal to the chromatic number. This is not surprising given the purely local nature of the algorithm.

Over shorter times, e.g., the first ten steps, the reduction in conflicts is best for moderate values of p, e.g., 0.3–0.5. Note that even though high p eventually results in degrees of conflict the same as, or even lower than, those for lower p, the short-term results for high p show that high p is ill-advised for real-time applications.

C. *Effect of Number of Colors*

The number of colors compared with the chromatic number gives one measure of how tightly constrained a problem is: if the number of colors is less than the chromatic number, then the problem is over-constrained and it is impossible to eliminate all conflicts; if the number of colors is equal to the chromatic number, then the problem is critically constrained; if the number of colors is greater than the chromatic number, then the problem is under-constrained.

The effect of constraint tightness is shown in Figure 5 for 2-dimensional Cartesian graphs with diagonals, as described above. (Note that each line, corresponding to different number of colors, is separately normalized.) In these results, when the problem is over-constrained, the algorithm converges to a near-optimal coloring ($\gamma = 0.509$ on average for 2 colors, compared with a theoretical minimum of 0.5, and 0.382 for 3 colors compared with a theoretical minimum of 0.375).

When the problem is critically constrained, the algorithm achieves a low, but non-zero, degree of conflict ($\gamma = 0.03$). When the problem is under-constrained, the algorithm reduces the number of conflicts to zero.

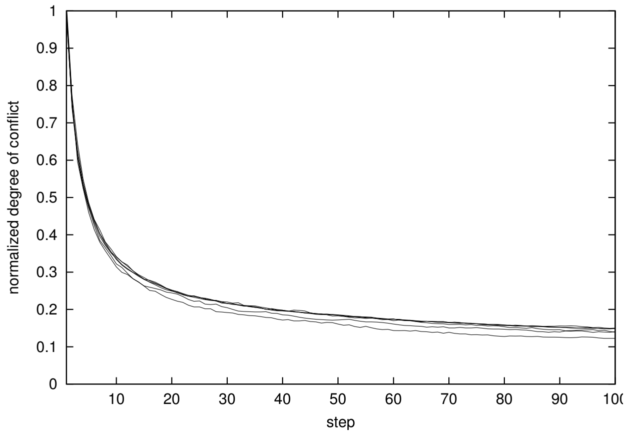

Fig. 6. Effect of graph size

D. *Scalability*

Figure 6 shows typical FP performance with 4 colors on 2-dimensional Cartesian graphs (with diagonals) of various sizes, ranging from 400 to 4000 nodes (each line is an average over 20 graphs of the same size). The effect of graph size (for large graphs) is small. There is likely no correlation between the convergence value of the degree of conflict and the graph size, but an appropriate statistical analysis has not been performed.

Examination of the algorithm shows that FP's per-node, per-step storage, computational and communication *costs* also do not depend on the graph size — these costs depend on (mean) degree of the graph and for the classes of problems under consideration, the degree of the graph is independent of the size of the graph. (For some well-known classes of graph, such as complete graphs, the graph's degree does depend on its size, but such classes are not likely to be important in the types of problem under consideration.)

E. *Robustness*

In these experiments, a graph of size N is initially constructed and some fraction q of the nodes (together with all

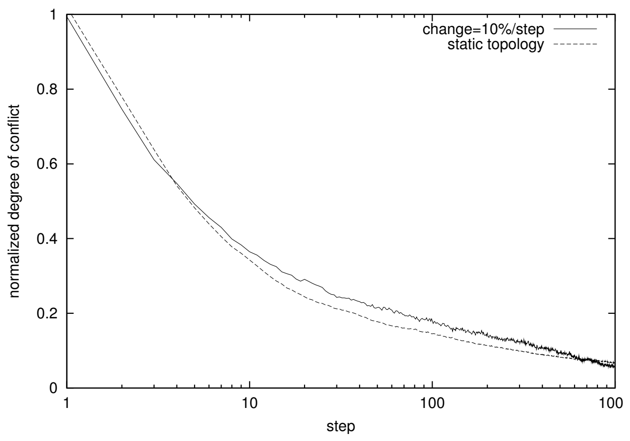

Fig. 7. Effect of continual, minor change

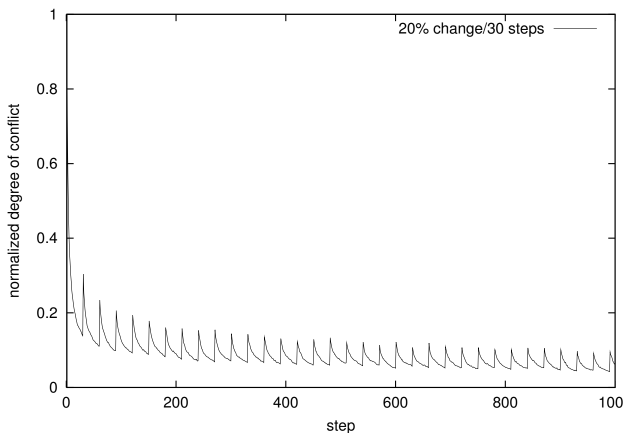

Fig. 8. Effect of intermittent, major change

incident edges) is randomly removed and recorded. The graph is then transformed every P steps as the coloring algorithm runs, as follows:

• A fraction q of the nodes (with all incident edges) is randomly removed and recorded.
• From the $2qN$ nodes that are currently removed, qN are randomly selected and reinserted into the graph (along with all removed edges for which both end nodes are now present in the graph).

Figure 7 shows the effect of small-scale changes occurring continually: the effect on the degree of conflict is small. Figure 8 shows the effect of large-scale changes occurring intermittently: the degree of conflict spikes immediately after a change, but quickly reduces again (and, in the long term, continues to reduce overall).

The most important result of these experiments is that they demonstrate that the algorithm does not suffer catastrophic failure, even when subject to quite drastic changes.

The effect of communication noise can also be demonstrated by subjecting each color-change message to a random process that may cause the message to be discarded or corrupted. The effect on the performance of the algorithm is incremental — small amounts of noise cause small increases in the degree of conflict, larger amounts of noise cause larger increases — but again, catastrophic failure is not observed.

V. Related Work

Most work on distributed constraint satisfaction or optimization addresses algorithms in the classes of simulated annealing or parallel branch-and-bound, in which *the search space* is distributed over some network of nodes and in which each node contains a complete solution to the problem.

In contrast, in the algorithm considered here, a single solution is distributed over the network of nodes, so that each node contains only a small fragment of a complete solution. This latter approach seems better suited to real-time control problems, in which communication latency essentially prohibits the gathering of information into a single node.

Other work on this latter form of distribution has been carried out by Yokoo *et al.* [8]. A deterministic version of the FP algorithm considered here was published by Fabiunke [3] — its performance is compared with the FP algorithm in [4]. However, these and other works seem to emphasize the long-term convergence properties of algorithms, whereas the property considered most important in this paper is the short-term improvement, since that relates directly to real-time responsiveness.

VI. Conclusion and Future Work

This paper describes a simple, peer-to-peer constraint optimization algorithm designed for sparsely-connected networks. In-depth experiments on graph coloring show the algorithm to be scalable and robust, and capable of long-term stability and good short-term dynamics when used appropriately. Proof-of-concept experiments on sensor coordination show that algorithm can be applied to realistic applications.

Further investigations should include the following:

• Other neighborhoods: the algorithm described here considers just the interaction of a node with its immediate neighbors. Other algorithms might consider the mutual interactions of those neighbors, or more distant nodes.
• Dynamic activation probabilities: the algorithm reported here uses a fixed, uniform activation probability. It seems plausible that dynamically adapting the activation probability, based on local information, would improve performance. However, initial experiments along these lines have yet to find significant improvements over the fixed probability algorithm.
• In-depth experiments on sensor networks: the performance of an FP-based resource manager could be compared with, say, the performance of a manager that selects measurements at random or based on a simple, local rule (one that does not allow for inter-sensor coordination). It should

also be possible to assess the performance of the manager directly, rather than through tracking performance: good tracking is of course the ultimate objective of the sensor network, but it can be difficult to separate the performance of the resource manager from the performance of the tracker.

• Experiments are currently underway on using the FP-based resource manager on physical sensors (rather than simulated sensors). Several tangential problems need to be resolved (e.g., accounting for physical effects such as multi-path interference, caused by reflections of the radar beam) before the performance of the distributed constraint optimization algorithm can be assessed.

REFERENCES

[1] *The ANTs Challenge Problem*,
http://ants.kestrel.edu/challenge-problem/index.html

[2] *Iterated Greedy Graph Coloring and the Difficulty Landscape*, Joseph Culberson, Technical Report TR 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992

[3] *Parallel Distributed Constraint Satisfaction*, Marko Fabiunke, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), pp. 1585-1591, Las Vegas, June 1999

[4] *An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs* Stephen Fitzpatrick & Lambert Meertens, 1^{st} Symposium on Stochastic Algorithms: Foundations and Applications, 13-14 December 2001 Berlin, Germany, Lecture Notes in Computer Science 2264, Kathleen Steinhofel (Ed.), Springer-Verlag ISBN 3-540-43025-3, pp. 49-64

[5] *Experiments with Parallel Graph Coloring Heuristics*, Gary Lewandowski & Anne Condon, in *Cliques, Coloring and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science , Volume 26, American Mathematical Society, 1996, pages 309-334

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

[7] *Workshops on Networked Embedded Systems Technology*:
http://www.dsic-web.net/ito/meetings/nest_mar2000/index.html

[8] *The Distributed Constraint Satisfaction Problem: Formalization and Algorithms*, Makoto Yokoo, Edmund H. Durfee, Toru Ishida & Kazuhiro Kuwabara, IEEE trans. on Knowledge and Data Engineering, vol. 10, no. 5, September/October 1998