Experiments on Dense Graphs with a Stochastic, Peer-to-Peer Colorer

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

Abstract

This paper reports on a simple, stochastic, scalable, peer-to-peer algorithm for aproximately solving distributed constraint problems in soft real time. The performance of the algorithm is assessed using k-colorings of random graphs having known chromatic number and edge probability ranging from moderate to high.

Introduction

Previous papers (Fitzpatrick & Meertens 2001; Meertens & Fitzpatrick 2001) have reported on a simple distributed algorithm for approximately solving large, sparse, distributed constraint problems. Such problems naturally arise in distributed resource management applications, with the constraint variables representing control states of physically separated resources.

A prototypical application is a large network of simple, localized, battery-powered sensors that communicate using high-latency radio communication to coordinate their measurements to achieve high-quality fusion of their data and to conserve energy by avoiding redundant measurements.

In the reported algorithm, each constraint variable is associated with a computational node that is solely responsible for assigning values to the variable. When a node assigns a value, it transmits the value to its neighbors: two nodes are neighbors iff their variables are neighbors, i.e., are connected by a constraint.

When determining what value to assign its variable, a node chooses a value that maximizes the number of constraints that it *believes* are satisfied, given what the node knows of the values of neighboring variables. The algorithm is iterative: each node periodically updates and transmits its value, receives new values from its neighbors, and updates and transmits its own value, and so on. The algorithm is scalable, in that a node's per-step costs are proportional to the number of neighbors rather than to the total number of variables.

The hope is that over time the number of constraints that are actually satisfied increases, and that perhaps eventually all of the constraints are satisfied. However, the algorithm is designed to quickly satisfy many constraints with low communication costs, rather than to ensure ultimate perfection; consequently, it is best thought of as an *anytime* algorithm that iteratively improves on a published solution.

Since communication is not instantaneous, it is possible that two neighboring nodes update their variables simultaneously, in which case the value that one thought the other had was incorrect. The use of such outdated information can lead to *incoherence*: neighbors' variable changes that are intended to be mutually beneficial turn out to be mutually detrimental.

Incoherence could be avoided by imposing a total order on the nodes, so that only one can update at any given time (and have sufficient time to communicate its new value to its neighbors). However, this is a sequential solution and is not scalable. Alternatively, a partial order could be imposed, so that *neighbors* are prohibited from simultaneous updates. This is equivalent to coloring the nodes, and so is likely just as hard a problem as the original constraint problem. (It also turns out that this is too strict a requirement: two neighbors are not *guaranteed* to cause mutual harm if they update simultaneously.)

In the reported algorithm, a stochastic approach is used to try to ensure sufficient coherence: in a particular time period, a randomly selected fraction of the nodes *activate*, and only those nodes that activate are allowed to change value. Each node determines its own activation status by comparing a randomly generated number with a fixed *activation probability*: the node activates iff the random number is less than the activation probability.

The activation probability thus provides a simple control for balancing parallelism aginst coherence: a low activation probability reduces the likelihood of neighbors simultaneously updating, but also reduces the degree of parallelism; the converse also holds.

In previous papers, the convergence and dynamics of this algorithm were investigated using k-colorings of *sparse* graphs; that is, graphs in which each node is connected to only a small fraction of the other nodes. In this paper, the

*This work is sponsored in part by DARPA through the 'Autonomous Negotiating Teams' program under contract #F30602-00-C-0014, 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.

investigation is extended to graphs having moderate to high density; i.e., in which each node is connected to a significant or large fraction of the other nodes.

The structure of the paper is as follows: a optimization version of graph coloring is defined, in which the objective is to minimize the number of edges that connect nodes of the same color; the algorithm is defined in detail; experimental results are presented for a range of graph densities and algorithm parameters; conclusions and related work are presented.

Soft Graph k-Coloring

Graph k-coloring is a prototypical constraint satisfaction problem. Each vertex in an undirected graph is to be assigned one of k colors (i.e., an integer in Z_k) such that neighboring vertices (i.e., vertices connected by an edge) have different colors.

The *quality* of a given coloring can be measured by the *degree of conflict* γ, defined as the fraction of edges that connect vertices having the same color; such an edge is refered to as a *conflict*. A degree of conflict of 1 corresponds to every vertex (in each connected component) having the same color; a degree of conflict of 0 corresponds to the classical notion of a *proper coloring*. The objective in *soft* graph coloring is to minimize the number of conflicts, or equivalently to minimize γ.

A random assignment of k colors to vertices has an expected degree of conflict of $1/k$. Since a random coloring can be computed in a distributed system with no communication, it provides a useful baseline for non-random colorers: it is to be hoped that they can achieve a better-than-random score. Consequently, it is useful to defined the *normalized* degree of conflict Γ by $\Gamma \equiv k\gamma$: a random coloring has an expected value of 1 using the normalized metric.

In the experiments reported here, the graphs are random graphs having bounded chromatic number, where the chromatic number is the smallest number of colors for which a proper coloring can be achieved. The graphs are constructed using the procedure shown in Figure 1, which ensures that there is at least one proper k-coloring for some given k, so that the graph's chromatic number cannot be higher than k. Moreover, if the mean degree is high, it is likely that the graph's chromatic number is equal to k.

The *edge probability* p_e is the ratio of the number of actual edges, $Nd/2$, to the number of possible edges, $N(N-1)/2$. If p_e is small, the graph is said to be sparse; if p_e is high (near 1), the graph is said to be dense.

The FP(α) Algorithm

The algorithm reported here is a distributed, synchronous algorithm. After random initialization, the nodes execute a synchronized cycle of stochastic activation (with probability α), choosing a color that is best (given what are believed to be the neighbors current colors), and transmitting any color change to neighbors — see Figure 2.

As shown, the algorithm requires two synchronization steps for every coloring step: this is mainly to emphasize

- The number of vertices, N, the mean degree, d, and the desired chromatic number, k, are given.
- A random assignment of colors to nodes is computed.
- The required number of edges, E, is computed from the given parameters: $E \equiv Nd/2$ (the factor of $1/2$ comes about because the edges are undirected and are counted in the degree of the nodes at both ends).
- Each of the E edges is constructed by randomly choosing two different and currently unconnected nodes, and connecting them with an edge if their colors are different.
- Once E edges have been constructed, the coloring is discarded.

Figure 1: Procedure for constructing random graphs

Each node i repeatedly executes the following cycle, in which c_i denotes i's color at the start of one iteration of the cycle and c_i' denotes the color chosen by i during the iteration:

1 Synchronize.

2 Compute a histogram, H_i, of the colors node i believes its neighbors currently have.

3 Generate a random number r_i, where $0 \leq r_i < 1$.

4a If some neighbor is believed to have the current color (i.e., $H_i(c_i) > 0$) and $r_i < \alpha$, choose any color c_i' that has a minimal value in H_i (i.e., that is believed to be least used among the neighbors).

4b Otherwise, do not change color ($c_i' = c_i$).

5 Synchronize.

6 If the color has been changed ($c_i' \neq c_i$), transmit the new color to the neighbors and adopt the new color as the current color.

Figure 2: The FP(α) algorithm, where $0 < \alpha \leq 1$

that a node's knowledge of its neighbors' colors has an unavoidable latency. In practice, synchronization is not critical; indeed, experiments have suggested that asynchronous operation can improve the convergence properties of the algorithm provided that the mean communication latency is less than half the mean period between color updates. In other words, the communication latency limits how frequently colors can be updated without introducing too much incoherence.

Note that in step 4a, there may be more than one color that is minimal in the histogram: one such color is chosen at random. It may happen that the current color is minimal in the histogram: it is treated the same as any other minimal color.

One of the conditions on step 4a, that the current color is believed to be in use by at least one neighbor, has been found to improve the performance of the algorithm for underconstrained problems (when the number of colors being used for the coloring is much higher than the chromatic number).

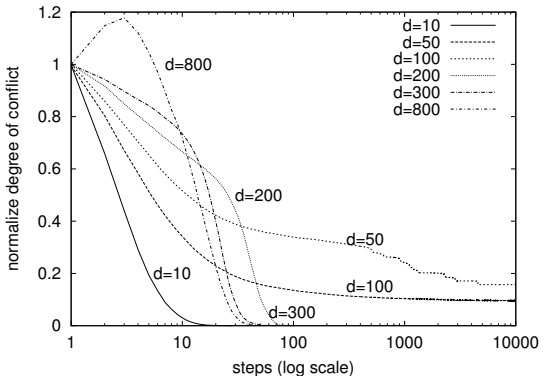

Figure 3: Γ over time, #colors=10, #nodes=900

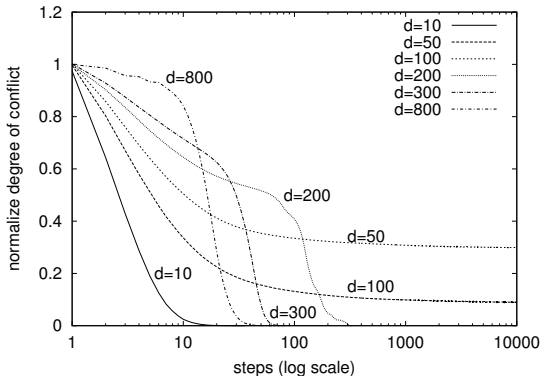

Figure 4: Γ over time, #colors=10, #nodes=2500

For other classes of problems, it is not believed to have much effect.

Experimental Results

Figure 3 shows experimental results for FP(0.2) on random graphs of 900 nodes and of chromatic number 10, using 10 colors. Results are shown for graphs having mean degrees $d = 10$, 50, 100, 200, 300 and 800, corresponding to edge probabilities of approximately 0.01, 0.06, 0.11, 0.22, 0.33 and 0.89. In each case, the results shown are averages over 20 colorings on randomly generated graphs.

The results for the low-density graphs ($d = 10$ to 100) are qualitatively in agreement with previous results on sparse graphs having regular structure (e.g., Cartesian grids with nodes at integer coordinates in a 2-dimensional plane and with edges between immediate neighbors along the axes and, in some cases, along the diagonals). The algorithm typically quickly reduces the number of conflicts and may produce a proper coloring, or may tend to a non-zero asymptote.

The suprising results are those for higher-density graphs ($d \geq 200$): the algorithm's initial reduction in conflicts is slower than for sparse graphs, but it soon precipitates a proper coloring. Note that initially the results for $d = 800$ are worse than what would be expected from a random coloring.

Qualitatively similar results are observed for larger graphs and for higher chromatic numbers. For example, Figure 4 shows results for graphs having 2500 nodes. Note that the results are not exactly the same: in particular, the initial performance for $d = 800$ is not as bad as for the smaller graphs: it may be speculated that the maximum activation probability that can be used without causing worse-than-random initial results is determined by the edge density, but not enough experimental data has been gathered yet to derive a precise relationship.

Nevertheless, it is informative to examine how the activation probability affects the overall performance. Figure 5 shows the *total* normalized degree of conflict accumulated over a run of 10000 steps; i.e., the sum of the per-step normalized degree of conflicts. This corresponds to the area

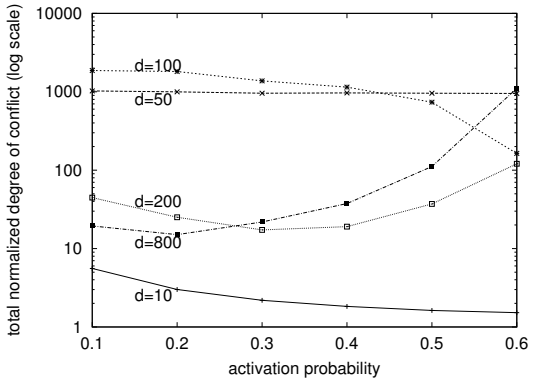

Figure 5: Total Γ over 10000 steps, #colors=10, #nodes=900

under the curves in the preceeding performance plots. (As before, the shown results are averages over 20 runs.)

For $d = 10$, high activation probabilities can profitably be used: the number of conflicts is quickly reduced to zero. For $d = 50$, the activation probability has little effect in these experiments; however, it should be noted that the algorithm does not achieve proper colorings for $d = 50$ in the time allowed and it may be that longer runs would exhibit different behaviors.

For $d = 100$, the lowest total conflicts is suprisingly achieved by an activation probability of 0.6: it would be expected that such a high activation probability would result in a high degree of incoherence. Indeed, Figure 6 shows that the quality of the coloring produced by FP(0.6) varies rapidly over time, suggesting that the behavior of FP(0.6) is quasi-chaotic. It would seem that the high degree of incoherence enables FP(0.6) to escape from local minima that trap more sedate colorers. In the context of a practical application, it may be possible to decide if the final result (a proper coloring) outweighs the poor intermediate results.

For dense graphs ($d \geq 200$), a moderate activation probablity, in the range 0.2-0.3, produces the fewest total conflicts. (Figure 5 shows results for only two degrees in this range — the other results are similar.) These values also produce reasonable short-term reductions in the number of con-

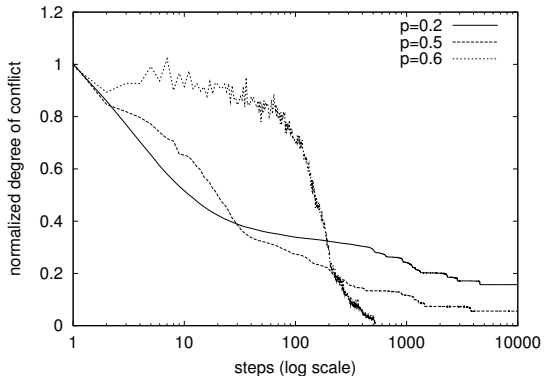

Figure 6: Γ over time, #colors=10, #nodes=900, mean degree=100

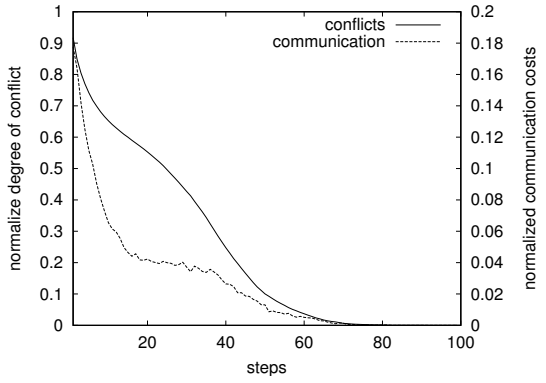

Figure 7: Γ and communication costs over time, #colors=10, #nodes=900, mean degree=200

flicts. These results are somewhat surprising — it might be expected that with, say, 200 neighbors, a node should have an activation probability of around 1/200 to avoid incoherence. An activation probability of 0.2 implies that around 40 of each node's neighbors are activating each step and managing to appropriately choose one of only 10 colors.

However, a node need not change color every time it activates. Figure 7 shows the conflicts and normalized communication costs over time for FP(0.2) on graphs with mean degree 200. The normalized communication cost (plotted against the right vertical axis) of a single step is the fraction of nodes that change color in that step — this is refered to as a 'communication' cost because color changes must be transmitted to neighbors.

The plot shows that initially a high number of nodes do change color, so the fact that they manage to achieve coherent results remains a surprise. It may be speculated that what is important is not how many nodes in a neighborhood activate simulataneously, but whether the two nodes at either end of a given edge activate simultaneously, since then incoherence might cause the edge to become a conflict. If this is the case, then it is the *rank* of the edge that determines the optimal activation probability (i.e., the number of nodes

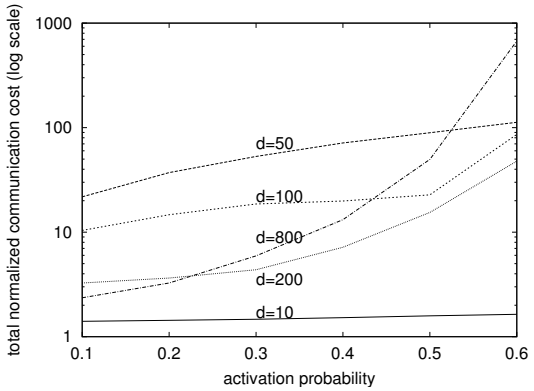

Figure 8: Total communication costs, #colors=10, #nodes=900

connected by a single edge). In the reported experiments, only binary edges were considered — ongoing experiments are investigating hyper-edges of higher rank.

Figure 8 summarizes the communication costs for various densities and activation probabilities. For sparse graphs, the costs increase approximately linearly with the activation probability. This seems reasonable: the more nodes activate, the more change color. (Note the jump in costs for $d = 100$ and $\alpha = 0.6$, corresponding to the quasi-chaotic behavior discussed above.)

For denser graphs, the costs increase super-linearly with the activation probability: the denser the graph, the more exagerated the growth. It is likely that this is caused by incipient quasi-chaotic behavior (as indicated by the worse-than-random initial performance shown in Figure 3).

Conclusions

The FP algorithm was initially designed to achieve good-enough approximate solutions to large, sparse, distributed constraint problems in soft real time. The experiments reported here indicate, quite surprisingly, that it may also be useful for dense problems, although some care is required in its use (e.g., if temporary quasi-chaotic behavior is undesirable).

Intermediate ranges of density may represent the most difficult problems for this algorithm: for intermediate densities, the algorithm reduces the number of conflicts significantly below random, but often fails to find a perfect solution; instead, it seems to asymptotically approach a solution that has a significant number of conflicts.

One surprising result of the reported experiments is that the activation probability need only be slightly reduced as the density increases dramatically: values in the range 0.2-0.3 seem to be generally useful.

The results presented here are initial. Further experiments should address the performance of the algorithm on higher-rank constraints and on a wider range of graphs (having high chromatic numbers). It would be useful to determine either experimentally or analytically such relationships as how the optimal activation probability depends on parameters of the graph.

Related Work

The Fixed Probability algorithm is an extension of a deterministic algorithm described in (Fabiunke 1999). Yokoo (Yokoo *et al.* 1998) 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 comparisons of the FP and DB algorithms is reported in (Zhang 2002).

References

Fabiunke, M. 1999. Parallel Distributed Constraint Satisfaction. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1585–1591. Las Vegas.

Fitzpatrick, S., and Meertens, L. 2001. An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs. In Steinhofel, K., ed., 1^{st} *Symposium on Stochastic Algorithms: Foundations and Applications*, number 2264 in Lecture Notes in Computer Science, 49–64. Springer-Verlag. Berlin, Germany.

Meertens, L., and Fitzpatrick, S. 2001. 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, 3260 Hillview Avenue, Palo Alto, California 94340, U.S.A.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE trans. on Knowledge and Data Engineering* 10(5).

Zhang, W. 2002. TBD.