In Proceedings of the International Symposium on Circuits and Systems, pages V-163-V-166, Sydney, Australia, May 2001.

ADAPTIVE NEGATIVE CYCLE DETECTION IN DYNAMIC GRAPHS

Nitin Chandrachoodan, Shuvra S. Bhattacharyya[†]*and K. J. Ray Liu*

Department of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742
(nitin,ssb,kjrliu@eng.umd.edu)

ABSTRACT

We examine the problem of detecting negative cycles in a dynamic graph, which is a fundamental problem that arises in electronic design automation and systems theory.

We introduce the concept of *adaptive* negative cycle detection, in which a graph changes over time, and negative cycle detection needs to be done periodically, but not necessarily after every individual change. Such scenarios arise, for example, during iterative design space exploration for hardware and software synthesis. We present an algorithm for this problem, called the *Adaptive Bellman-Ford (ABF)* algorithm, based on a novel extension of the well known Bellman-Ford algorithm. The ABF algorithm allows us to systematically adapt information for a given graph to a modified version of the graph. We show that the ABF algorithm significantly outperforms previously available approaches for dynamic graphs, which either recompute negative cycle information from scratch whenever a graph is modified, or process the modifications one at a time ("incrementally").

As an application of the ABF technique, we show that it can be used to obtain a very fast implementation of Lawler's technique for the computation of the maximum-cycle mean (MCM) of a graph, especially for a certain important kind of *sparse graph.* We further illustrate the application of the ABF technique to design-space exploration by developing automated search techniques for scheduling iterative data-flow graphs.

1. INTRODUCTION

Several problems in circuits and systems theory require the solving of constraint equations [1, 2, 3]. Examples include VLSI layout compaction, computing maximum operational speed of circuits, and performance analysis of interactive (reactive) systems. Several problems of interest actually consist of the special case of difference constraints (each constraint expresses the minimum or maximum value that the difference of two variables in the system can take). These problems can be attacked by faster techniques than linear programming, such as by solving a shortest path problem on a weighted directed graph. A related problem is the detection of negative cycles in the graph, which would indicate in-feasibility of the corresponding constraint system. Considerable effort has been spent on finding efficient algorithms for negative cycle detection. Cherkassky and Goldberg [1] have performed a comprehensive survey of existing techniques. Their study shows some interesting features of the available algorithms, *e.g.* for a large class of random graphs, the worst case performance bound is far more pessimistic than the observed performance.

There are also situations in which it is useful or necessary to maintain a feasible solution to a set of difference constraints as a system evolves. Typical examples of this would be real-time or interactive systems, where constraints are added or removed one (or several) at a time, and after some modifications, it is required to determine whether the resulting system has a feasible solution and if so, to find it. In these situations, it is often more efficient to adapt existing information to aid the solution of the constraint system. In the area of computer-aided system design, it is often easy to cast the problem of design space exploration in a way that benefits from this approach.

Several researchers[4, 5, 6] have worked on the area of *incremental computation.* They have presented analyses of algorithms for the shortest path problem and negative cycle detection in *dynamic* graphs. These approaches have focussed on the incremental (single change) problem and cannot be extended to take advantage of multiple changes being made at the same time.

In this paper, we present an approach which generalizes the adaptive approach beyond single increments: we address multiple changes being made to the graph simultaneously. This solution is based on enhancing the Bellman-Ford algorithm for shortest paths, and is, to our knowledge, the first algorithm to attack the multiple change problem. We present simulation results comparing our method against the single-increment algorithm proposed in [2].

To illustrate the advantages of our adaptive approach, we present some sample applications requiring the solving of difference constraint problems, which therefore benefit from the application of our technique. We show how the ABF technique can be used to derive a fast implementation of Lawler's algorithm for the problem of computing the maximum cycle-mean (MCM) of a weighted directed graph. We present experimental results comparing this against Howard's algorithm [7, 3], which appears to be the fastest algorithm available in practice. We find that for graph sizes and node-degrees similar to those of real circuits, the ABF-based algorithm often outperforms Howard's algorithm.

We also present a search technique to compute efficient schedules for iterative dataflow graphs, that uses the adaptive negative cycle detection algorithm as a subroutine. We illustrate the use of this local search technique by applying it to a problem from high-level synthesis, namely resource constrained scheduling for minimum power in the presence of functional units that can operate at multiple voltages.

In Section 2, we present our enhancements to the Bellman-Ford algorithm that enable it to work on multiple changes to a graph efficiently. Section 3 compares our algorithm against exist-

[*]This research was supported in part by the US National Science Foundation Grant #9734275 and NSF NYI Award MIP9457397

[†]Also with the University of Maryland Institute for Advanced Computer Studies.

Algorithm 1 Adaptive Bellman-Ford Algorithm

```
Require: Graph $G(V, E), dist(v), weight(e)$
Ensure: updated $dist(v)$ such that $\forall e = (u \rightarrow v) \in E : dist(v) - dist(u) \leq weight(u \rightarrow v)$
 1: $Q1 \leftarrow \phi, Q2 \leftarrow \phi$
 2: for all $e \in E$ do
 3:   if $dist(v) - dist(u) > weight(u \rightarrow v)$ then
 4:     append $u$ to $Q1$
 5:   end if
 6: end for
 7: while $Q1$ not empty do
 8:   $u \leftarrow pop(Q1)$
 9:   for all $v$ adjacent to $u$ in $G$ do
10:     if $dist(v) - dist(u) > weight(u \rightarrow v)$ then
11:       delete subtree rooted at $v$
12:       if $u$ was in the subtree deleted above then
13:         negative cycle detected: return
14:       else
15:         make $v$ a child of $u$ {constructing subtree}
16:         append $v$ to $Q2$
17:       end if
18:     end if
19:   end for
20:   if $Q2$ is empty then
21:     return {completed: $dist$ satisfies constraints}
22:   else
23:     $Q1 \leftarrow Q2, Q2 \leftarrow \phi$
24:   end if
25: end while
```

ing alternatives. Section 4 then gives details of the applications mentioned above, and presents some experimental results. Finally, we present our conclusions and examine areas that would be suitable for further investigation.

2. THE ADAPTIVE BELLMAN-FORD ALGORITHM

In this section, we propose our extensions to the Bellman-Ford algorithm which allow us to handle multiple changes adaptively.

We first note that the problem of detecting negative cycles in a weighted directed graph (digraph) is equivalent to finding whether or not a set of difference inequality constraints has a feasible solution. To see this, observe that if we have a set of difference constraints of the form $x_i - x_j \leq b_{ij}$ we can construct a digraph with vertices corresponding to the x_i, and an edge (e_{ij}) directed from the vertex corresponding to x_i to the vertex for x_j such that $weight(e_{ij}) = b_{ij}$. This procedure is performed for each constraint in the system and a weighted directed graph is obtained. Solving for shortest paths in this graph would yield a set of distances $dist$ that satisfy the constraints on x_i. This graph is henceforth referred to as the *constraint graph.*

The usual technique used to solve for $dist$ is to introduce an imaginary vertex s_0 to act as a source, and introduce edges of zero-weight from this vertex to each of the other vertices. In this way, we can use a single-source shortest paths algorithm to find $dist$ from s_0, and any negative cycles (infeasible solution) will occur only in the original graph, since the new vertex and edges cannot create cycles. This graph is referred to as the *augmented graph* [2].

The algorithm for adaptive negative cycle detection presented in Alg. 1 is an adaptation of Tarjan's subtree disassembly method for negative cycle detection in static graphs [1]. We henceforth refer to this modified algorithm as the "Adaptive Bellman-Ford algorithm" or ABF algorithm, to stress that it adapts a solution to the original graph to obtain the solution to the constraints corresponding to the altered graph.

The enhancements to the basic algorithm are in lines 2-6. This code initializes the set of active vertices to those involved in a constraint violation. By retaining information across calls to the routine, the computation here can be much less than when we start from scratch. After this, the normal operation of the Bellman-Ford algorithm follows. Because of the generality of the idea, other algorithms can also be used with the adaptive enhancements. Also, in most applications, it is possible for the higher level application to pass information to the routine that helps it to find those edges where a change has occurred, thus further saving some computation, though this will not change the overall complexity of the algorithm (the complexity is dominated by the actual negative cycle detection computation).

3. COMPARISON AGAINST OTHER APPROACHES FOR DYNAMIC GRAPHS

We compare the ABF algorithm against (a) the incremental algorithm developed in [2] for maintaining a solution to a set of difference constraints (referred to here as the RSJM algorithm), and (b) a modification of Howard's algorithm [7], since it appears to be the fastest known algorithm to compute the cycle mean, and hence can also be used to check for feasibility of a system. Our modification allows us to use the adaptive technique to reduce the computation in this algorithm.

Note that the RSJM algorithm from [2] uses Dijkstra's algorithm as its core routine. This implies that it cannot in principle be extended to handle multiple changes in the way that we have done with the Bellman-Ford algorithm. We have used the implementation of Howard's algorithm from the authors of [7], and have taken into account the modifications suggested by Dasdan [3].

We have restricted our attention to *sparse* graphs, or *bounded degree* graphs. In particular, we have tried to keep the vertex-to-edge ratio similar to what we may find in practice, as in, for example, the ISCAS benchmarks. Such graphs are relevant because real circuits tend to have properties like bounded fanout and small numbers of inputs, which result in graphs of small bounded degree.

We have implemented all the algorithms under the LEDA [8] framework for uniformity. The tests were run on random graphs, with several random variations performed on them thereafter. A "change" to the graph under consideration involved either addition or deletion of an edge, or changing the weight of an existing edge. We refer to such changes as "edge-change operations". Note that modifying a vertex can affect several edges. It is because of this that we restricted our focus to edge changes so that we could control the number of changes. We use the term "batch-size" to refer to the number of changes in a multiple change update. This is a useful parameter to understand the performance of the algorithms.

Figure 1 shows a comparison of the running time of the 3 algorithms on random graphs. The graphs in question were randomly generated and had 1,000 vertices and 2,000 edges each. A sequence of 10,000 edge change operations (as defined above) was applied to each of them. Each point in the plot corresponds to an average over 10 runs using randomly generated graphs. The X-axis shows the batch-size ("granularity" of changes). Note that

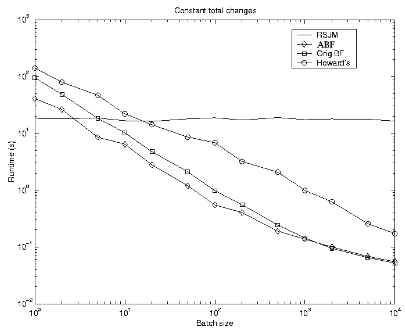

Figure 1: Comparison of algorithms as batch size varies

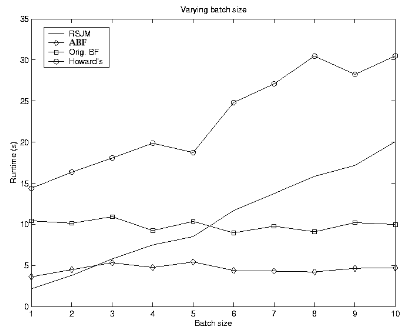

Figure 2: Constant number of iterations at different batch sizes

the delayed update feature is not used by algorithm `RSJM`, which uses the fact that only one change occurs per test to look for negative cycles. As can be seen, the algorithms that use the adaptive modifications benefit greatly as the batch size is increased, and even among these, the `ABF` algorithm far outperforms Howard's algorithm. This is because the latter actually performs most of the computation required to re-compute the maximum cycle-mean from scratch, which is far more than necessary.

Figure 2 shows a plot of what happens when we apply 1000 batches of changes to the graph, but alter the number of changes per batch, so that the total number of changes actually varies from 1000 to 100,000. As expected, `RSJM` takes total time proportional to the number of changes. But the other algorithms take nearly constant time as the batch size varies. The overall performance for the adaptive algorithm is dominated by overhead corresponding to the bookkeeping operations. By reducing the number of updates in the adaptive computation, the run-time for the adaptive algorithm is almost constant. However, as may be expected, as the batch-size increases asymptotically, the run-time of the ABF algorithm also starts to increase, and when the number of changes is of the order of the number of edges, it becomes equivalent to computing negative cycles from scratch.

4. APPLICATIONS

In this section, we present two applications that make extensive use of algorithms for negative-cycle detection. In addition, these applications also present situations where we encounter the same dynamic graph with sequences of small modifications – either in the edge-weights (sec. 4.1) or in the actual addition and deletion

Benchmark	$\frac{\lvert E\rvert}{\lvert V\rvert}$	$\frac{\lvert D\rvert}{\lvert V\rvert}$	orig.BF MCM	ABF MCM	Howard's algo.
s38417	1.416	0.069	2.71	0.29	0.66
s38584	1.665	0.069	2.66	0.63	0.59
s35932	1.701	0.097	1.79	0.37	0.09
s15850	1.380	0.057	1.47	0.18	0.36
s13207	1.382	0.077	0.73	0.12	0.35
s9234	1.408	0.039	0.57	0.06	0.11

Table 1: Run-time for MCM computation for the 6 largest ISCAS 89/93 benchmarks.

of a small number of edges (sec. 4.2). As a result, these provide good examples of the type of applications that would benefit from the adaptive solution to the negative cycle detection problem.

4.1. Maximum Cycle Mean computation

The first application we consider is the computation of the Maximum Cycle-Mean (MCM) of a weighted digraph. This is defined as the maximum over all directed cycles of the sum of the arc weights divided by the number of "delay" elements on the arcs. This metric plays an important role in discrete systems and embedded systems [3, 9], since it represents the greatest throughput that can be extracted from the system. There are also situations where it may be desirable to recompute this measure several times on closely related graphs, for example for the purpose of design space exploration. The first extensive study of algorithmic alternatives for this problem has been undertaken by Dasdan *et al.* [3]. This study concluded that the best existing algorithm in practice for this problem appears to be Howard's algorithm, which, unfortunately, does not have a known polynomial bound on its running time.

To model this application, the edge weights on our graph are obtained from the equation $weight(u \to v) = delay(e) \times P - exec\_time(u)$ where $weight(e)$ refers to the weight of the edge $e : u \to v$, $delay(e)$ refers to the number of delay elements (flip-flops) on the edge, $exec\_time(u)$ is the propagation delay of the circuit element that is represented by the vertex, and P is the desired clock period that we are testing the system for. In other words, if the graph with weights as mentioned above does not have negative cycles, then P is a feasible clock for the system. We can perform a binary search in order to compute P to any precision we require. This approach is attributed to Lawler [10].

Table 1 shows the run-times of the algorithms on some of the ISCAS 89/93 benchmark circuits. In the table, $|V|$ is the number of vertices in the graph, $|E|$ is the number of edges, and $|D|$ is the number of delay elements. The results clearly show that in several cases, Lawler's algorithm using our adaptive negative cycle detection technique outperforms even Howard's algorithm. We have conducted a larger analysis of the relative performance on random graphs [9], and the results show that for a large class of sparse graphs, the ABF-based algorithm is superior.

4.2. Search Techniques for Scheduling

A *schedule* of a dataflow graph on processors consists of an ordering of the vertices on the processors. If the resulting graph weighted with the execution times of the actors does not contain negative cycles as in section 4.1, a valid schedule has been found.

Example	Res. (5V+, 3.3V+,5V*)	T	Power saved	
			S and R	ABF
5th-order ellip.filt.	{2, 2, 2}	25	31.54%	29.88%
	{2, 1, 2}	25	18.26%	16.6%
	{2, 2, 2}	22	23.24%	24.9%
	{2, 1, 2}	21	13.28%	13.28%
FIR filt.	{1, 2, 1}	15	29.45%	34.36%
	{1, 2, 2}	10	17.18%	24.54%

Table 2: Comparison between ABF-based search and algorithm of Sarrafzadeh and Raje [11]

Since the underlying graph is always the same and only the vertex ordering imposes new edges, we can easily use adaptive negative cycle detection to perform the constraint checking efficiently.

We have examined the scheduling formulation addressed by Sarrafzadeh and Raje [11] to illustrate this. The problem is to schedule the graphs on a fixed architecture where a 5V adder takes 1 unit of time to execute, a 2V adder takes 2 units, and a multiplier takes either 1 unit or 2 units depending on the application. It is assumed that power consumed is proportional to V^2, so scheduling vertices on a 3.3V resource reduces the power consumption. We have attacked this problem by first scheduling each actor on its own resource, and then iteratively increasing resource sharing while maintaining a valid schedule till the resource constraint is met.

We have used only this schedule modification method, namely migrating vertices across processors. Further refinements could take into account the influence that a very basic search would have and try to implement some look-ahead. Already, the results match and even outperform that obtained in [11]. In addition, the ABF-based method has the benefit that it can handle any number of voltages/processors, and can also easily be extended to other problems, such as homogeneous processor scheduling [12]. Table 2 shows the power-savings that were obtained on the examples from [11]. S and R power saving indicates the power savings quoted in [11], while ABF power savings refers to the results obtained using the ABF-based scheduling approach. T is the overall timing constraint (the maximum iteration period bound that we are aiming for).

5. CONCLUSIONS

We have introduced an adaptive approach (the ABF algorithm) to negative cycle detection in dynamically changing graphs. Our technique explicitly addresses the common, practical scenario in which negative cycle detection must be periodically performed after intervals in which a small number of changes are made to the graph. We have shown by experiments that for reasonable sized graphs (10,000 vertices and 20,000 edges) our algorithm outperforms the incremental algorithm described in [2] even for changes made in groups of as little as 4-5 at a time.

We have also shown how our adaptive approach to negative cycle detection can be exploited to compute the maximum cycle mean of a weighted digraph, which is a relevant metric for many problems in the design and analysis of circuits and systems. We have compared our ABF-based MCM computation technique against the most efficient alternative, which is Howard's algorithm. We have shown that the ABF-based technique outperforms Howard's algorithm for sparse graphs which are commonly found in real circuits.

Since computing power is cheaply available now, it is increasingly worthwhile to employ extensive search techniques for solving NP-hard design problems such as scheduling. An efficient adaptive negative cycle detection algorithm can make this process more profitable. We have demonstrated this by employing our ABF algorithm within the framework of a search strategy for multiple voltage scheduling.

6. REFERENCES

[1] B. Cherkassky and A. V. Goldberg, "Negative cycle detection algorithms," Tech. Rep. tr-96-029, NEC Research Institute, Inc., March 1996.

[2] G. Ramalingam, J. Song, L. Joskowicz, and R. E. Miller, "Solving systems of difference constraints incrementally," *Algorithmica*, vol. 23, pp. 261–275, 1999.

[3] A. Dasdan, S. S. Irani, and R. K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems," in *36th Design Automation Conference*, pp. 37–42, ACM/IEEE, 1999.

[4] G. Ramalingam, *Bounded Incremental Computation.* PhD thesis, University of Wisconsin, Madison, August 1993. Revised version published by Springer Verlag (1996) as Lecture Notes in Computer Science 1089.

[5] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic shortest paths and negative cycle detection on digraphs with arbitrary arc weights," in *ESA98*, vol. 1461 of *Lecture Notes in Computer Science*, (Venice, Italy), pp. 320–331, Springer, August 1998.

[6] B. Alpern, R. Hoover, B. K. Rosen, P. F. Sweeney, and F. K. Zadeck, "Incremental evaluation of computational circuits," in *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pp. 32–42, 1990.

[7] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," in *Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[8] K. Mehlhorn and S. Näher, "LEDA: A platform for combinatorial and geometric computing," *Communications of the ACM*, vol. 38, no. 1, pp. 96–102, 1995.

[9] N. Chandrachoodan, S. S. Bhattacharyya, and K. J. R. Liu, "Negative cycle detection in dynamic graphs," Tech. Rep. UMIACS-TR-99-59, University of Maryland Institute for Advanced Computer Studies, September 1999. *http://www.cs.umd.edu/TRs/TRumiacs.html*.

[10] E. Lawler, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rhinehart and Winston, 1976.

[11] M. Sarrafzadeh and S. Raje, "Scheduling with multiple voltages under resource constraints," in *Proc. ISCAS 99*, 1999.

[12] S. M. H. de Groot, S. H. Gerez, and O. E. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," *IEEE Transactions on Circuits and Systems - I*, vol. 39, pp. 351–364, May 1992.