Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems

Ali Dasdan

Dept. of Computer Science
University of Illinois, Urbana, IL 61801
dasdan@cs.uiuc.edu

Sandy S. Irani and Rajesh K. Gupta

Dept. of Information and Computer Science
University of California, Irvine, CA 92697
{irani,rgupta}@ics.uci.edu

Abstract

The goal of this paper is to identify the most efficient algorithms for the optimum mean cycle and optimum cost to time ratio problems and compare them with the popular ones in the CAD community. These problems have numerous important applications in CAD, graph theory, discrete event system theory, and manufacturing systems. In particular, they are fundamental to the performance analysis of digital systems such as synchronous, asynchronous, dataflow, and embedded real-time systems. For instance, algorithms for these problems are used to compute the cycle period of any cyclic digital system. Without loss of generality, we discuss these algorithms in the context of the minimum mean cycle problem (MCMP). We performed a comprehensive experimental study of ten leading algorithms for MCMP. We programmed these algorithms uniformly and efficiently. We systematically compared them on a test suite composed of random graphs as well as benchmark circuits. Above all, our results provide important insight into the performance of these algorithms in practice. One of the most surprising results of this paper is that Howard's algorithm, known in the stochastic control community only, is by far the fastest algorithm on our test suite although the only known bound on its running time is exponential.

1 Introduction

1.1 Notation and definitions

Consider a digraph $G = (V, E)$ with n nodes and m arcs. Associate with each arc e in E two numbers: a weight (or cost) $w(e)$ and a transit time $t(e)$. The weight and transit time of a path in G is equal to the sum of the weights and transit times of the arcs on the path, respectively. The length of a path is equal to the number of arcs on the path. Let $w(C)$, $t(C)$, and $|C|$ denote the weight, transit time, and length of a cycle C in G.

The *(cycle) ratio* $\rho(C)$ and the *(cycle) mean* $\lambda(C)$ of cycle C are defined as

$$\rho(C) = \frac{w(C)}{t(C)},\ \ t(C) > 0,\ \ \text{and}\ \ \lambda(C) = \frac{w(C)}{t(C)} = \frac{w(C)}{|C|},$$

respectively. Note that $\rho(C)$ gives the average weight per transit time, and $\lambda(C)$ gives the average arc weight on C. The cycle ratio is historically called the cost to time ratio. The mean of C is a special case of its ratio in that its mean is obtained from its ratio by setting the transit time of every arc on C to unity.

The *minimum cycle ratio* (MCR) ρ^* and the minimum cycle mean (MCM) λ^* of G are defined as

$$\rho^* = \min_{C \in G}\{\rho(C)\}\ \ \text{and}\ \ \lambda^* = \min_{C \in G}\{\lambda(C)\},$$

respectively. In both cases, C ranges over all the cycles in G. The respective problems are called the *minimum cycle ratio problem* (MCRP) and the *minimum mean cycle problem* (MCMP). The definitions of the maximum versions of both of these problems are analogous.

1.2 Applications

The applications of both MCRP and MCMP are important and numerous. See [12] for the applications in graph theory. Our focus is on the applications in the CAD of digital systems. These problems have *fundamental* importance to the performance analysis of discrete event systems [3], which can also model digital systems. These problems are applicable to the performance analysis of such digital systems as synchronous [23], asynchronous [4], DSP [14], or embedded real-time [18]. Simply put, the algorithms for these problems are essential tools to find the cycle period of a given cyclic discrete event system. Once determined, the cycle period is used to describe the behavior of the system analytically over an infinite time period. For instance, the algorithms for these problems are used to compute the iteration bound of a dataflow graph [14], the time separation between event occurrences in a cyclic event graph [13], and the optimal clock schedules for circuits [22].

1.3 Related work

There are many algorithms proposed for both MCRP and MCMP. We give a comprehensive classification of the fastest and the most common ones in Table 1. References to a few old algorithms can be found in [12]. Note that as MCMP is a special case of MCRP, any algorithm for the latter problem can be used to solve the former problem. Conversely,

To appear in Proc. 36th Design Automation Conf. (DAC), Jun. 1999.

it is also possible to solve MCRP using an algorithm for MCMP [11]. Thus, we will focus only on MCMP in this paper.

In Table 1, the polynomial and pseudopolynomial algorithms are respectively ordered according to their worst-case running times. Those with the same running time are presented in alphabetical order of their inventors' names. Some references are cited more than once because they contain more than one algorithm. Some algorithms cannot produce the exact MCM or MCR, in which case they are said to return approximate results. The amount of error that can be tolerated in the approximate results can usually be controlled. This amount of error is denoted by ϵ, and called the precision of the algorithm.

1.4 Motivations, contributions, and methodology

Our main goal in this paper is to identify the most efficient algorithms for MCMP and MCRP and compare them with the current practice in the CAD community for solving similar problems. We have realized this goal with the following motivations and contributions:

(1) Despite the importance of MCMP and MCRP, most of the works in the CAD community that deal with these problems are not aware of most of the algorithms in Table 1. The most popular algorithms in the CAD community are Karp's algorithm [15] and Lawler's algorithm [17] and to some extent, Burns' algorithm [4]. We show that there are far more efficient algorithms than these popular ones. In particular, we show that Howard's algorithm is significantly faster than all the others. We provide an improved version of this algorithm. In [9], we also provide two bounds on its running time that are stronger than the previous bound.

(2) Most of the earlier works do not present any experimental analysis of even the algorithms that they introduce. There has not been a clear understanding of the performance of any of the algorithms in this paper although theoretical bounds on their running times have been proven. Finding more efficient implementation of these algorithms is very important because their applications usually need many runs of them, e.g., see [8]. This paper is the first study that systematically compares their performance and provides a great deal of insight into their behavior. We also provide some implementational improvements on most of the algorithms.

In this study, we have done a comprehensive empirical study comparing the ten leading MCM algorithms and the MCM versions of the MCR algorithms from Table 1, all of which are named in the table. The remaining algorithms in this table are not included in our study because they are very similar to the chosen ones. We implemented each algorithm in a uniform and efficient manner. We tested them on a series of random graphs, obtained using one generator from [5], and real benchmark circuits, obtained from logic synthesis benchmarks. The running time as well as representative operation counts, as advocated in [2], are measured and compared. We now give a review of the these algorithms and then present the experimental results and our observations.

2 Algorithms for MCMP

Below, we give a different formulation of MCMP and review the algorithms that we included in this study. More detailed discussion of these algorithms together with their pseudocode is given in [9]. This section is mainly for the convenience of the readers.

2.1 Definitions

The minimum cycle mean λ^* of a graph $G = (V, E)$ can be defined as the optimum value of λ in the following linear program:

$$\max\ \lambda\ \text{ s.t. }\ d(v) - d(u) \leq w(u,v) - \lambda,\quad \forall (u,v) \in E, \quad (1)$$

where $d(v)$ is called the *distance* (or the node potential) of v. When the inequalities are all satisfied, $d(v)$ is equal to the weight of the shortest path from s to v in G when λ^* is subtracted from every arc weight. The node s is arbitrarily chosen as the *source* in advance. Let G_λ denote the graph obtained from G by subtracting λ from its every arc weight. The minimum cycle mean λ^* is the largest value of λ for which G_λ has no negative cycles.

We say that an arc $(u, v) \in E$ is *critical* if $d(v) - d(u) = w(u, v) - \lambda$, which we refer to as the *criticality criterion*. We say that a node is *critical* if it is adjacent to a critical arc, and that a graph is *critical* if its every arc is critical. The critical subgraph of G_{λ^*} contains all the minimum mean cycles of G, as implied by Equation 1. The critical subgraph is important to computed because the critical subgraph of a graph G contains all the arcs and nodes that determine the performance of the system modeled by G. After running a MCM or MCR algorithm on G, the critical subgraph of G can easily be computed using its definition. As a result, we present each algorithm in the context of computing λ^* only.

We assume that the input graph G to the algorithm in the context is cyclic and strongly connected. This assumption simplifies most of the algorithms and generally improves their running times in practice. Note that if G is not strongly connected, its minimum cycle mean can be found easily: first partition G into its strongly connected components, run the algorithm on each strongly connected component, and then take as the minimum cycle mean of G the minimum of the cycle means returned by the algorithm. This is the way we implemented all of the algorithms.

2.2 Burns' algorithm

Burns' algorithm [4] is actually the minimum mean cycle version of the original Burns' algorithm for MCRP. We have discovered that the algorithm in [7] is identical to Burns' algorithm. Burns' algorithm is based on linear programming. It is an iterative algorithm constructed by applying the primal-dual method. It solves the above linear program (Equation 1) and its dual simultaneously. In essence, the behavior of Burns' algorithm is very similar to that of the parametric shortest path algorithms below such as the KO algorithm: The KO algorithm improves upon an initial acyclic critical subgraph of G until the critical subgraph becomes cyclic, at which point the minimum cycle mean is found. Burns' algorithm also operates on the critical subgraph and terminates when it becomes cyclic but every iteration, it reconstructs the critical subgraph from scratch.

2.3 Karp's algorithm and its variants

Define $D_k(v)$ to be the weight of the shortest path of length k from s, the source, to v; if no such path exists, then $D_k(v) =$

Table 1: Minimum mean cycle and minimum cost to time ratio algorithms for a graph G with n nodes and m arcs. (W, the max arc weight; T, the total transit time of G; N is the product of the out-degrees of all the nodes in G.)

	Name	Source	Year	Running time	Result	Complexity
			Minimum mean cycle algorithms			
1	DG	Dasdan & Gupta [8]	1997	$O(nm)$	Exact	Polynomial
2	HO	Hartmann & Orlin [12]	1993	$O(nm)$	Exact	Polynomial
3	Karp's	Karp [15]	1978	$\Theta(nm)$	Exact	Polynomial
4		Hartmann & Orlin [12]	1993	$O(nm + n^2 \lg n)$	Exact	Polynomial
5	YTO	Young, Tarjan, & Orlin [25]	1991	$O(nm + n^2 \lg n)$	Exact	Polynomial
6		Karp & Orlin [16]	1981	$\Theta(n^3)$	Exact	Polynomial
7	KO	Karp & Orlin [16]	1981	$O(nm \lg n)$	Exact	Polynomial
8	OA1	Orlin & Ahuja [21]	1992	$O(\sqrt{n}m \lg(nW))$	Approximate	Pseudopoly.
9	OA2	Orlin & Ahuja [21]	1992	$O(\sqrt{n}m \lg^2(nW))$	Approximate	Pseudopoly.
10		Cuninghame-Green & Yixun [7]	1996	$O(n^4)$	Exact	Polynomial
			Minimum cost to time ratio algorithms			
	Name	Source	Year	Running time	Result	Complexity
11	Burns'	Burns [4]	1991	$O(n^2 m)$	Exact	Polynomial
12		Megiddo [19]	1979	$O(n^2 m \lg n)$	Exact	Polynomial
13		Hartmann & Orlin [12]	1993	$O(Tm)$	Exact	Pseudopoly.
14	Lawler's	Lawler [17]	1976	$O(nm \lg(nW))$	Approximate	Pseudopoly.
15		Ito & Parhi [14]	1995	$O(Tm + T^3)$	Exact	Pseudopoly.
16		Gerez et al. [10]	1992	$O(Tm + T^3 \lg T)$	Approximate	Pseudopoly.
17		Gerez et al. [10]	1992	$O(Tm + T^4)$	Exact	Pseudopoly.
18	Howard's	Cochet-Terrasson et al. [6]	1997	$O(Nm)$	Exact	Pseudopoly.

$+\infty$. Karp's algorithm [15] is based on his observation that

$$\lambda^* = \min_{v \in V} \max_{0 \le k \le n-1} \frac{D_n(v) - D_k(v)}{n-k},$$

which is called Karp's theorem. Karp's algorithm computes each $D_k(v)$ by the recurrence

$$D_k(v) = \min_{(u,v) \in E} \{D_{k-1}(u) + w(u,v)\}, \;\; k = 1, 2, \ldots, n,$$
$$D_0(s) = 0, D_0(v) = +\infty, \;\; v \neq s.$$

Note that $d(v)$ and $D(v)$ are related to each other by the equation $d(v) = \min_{0 \le k \le n-1}\{D_k(v) - k\lambda\}$. As observed in [8, 12, 25], this recurrence, which is not implemented recursively, makes the best and worst cases of Karp's algorithm the same, which is why it runs in $\Theta(nm)$.

We have three improvements on Karp's algorithm: the DG algorithm, the HO algorithm, and the Karp2 algorithm. The DG algorithm [8] improves upon Karp's algorithm by eliminating unnecessary work introduced by the above recurrence. It works in a breadth-first manner in that starting from the source, it visits the successors of nodes rather than their predecessors, as done in the recurrence. This process creates an unfolding of G, and when the algorithm is implemented using linked lists, its running time becomes equal to the size of the "unfolded" graph. Depending on the structure of G, the running time ranges from $\Theta(m)$ to $O(mn)$.

The HO algorithm [12] also improves upon Karp's algorithm. It helps to terminate Karp's algorithm early without changing its structure, i.e., it still uses the above recurrence. It is based on the observation that many of the shortest paths computed as Karp's algorithm runs will contain cycles. If one of these cycles is critical, then the minimum cycle mean is found, which suffices to terminate the algorithm. The HO algorithm essentially checks the criticality of each cycle on the shortest paths computed. If the early termination is not possible, this algorithm can add an overhead of $O(n^2 + m \lg n)$ in total to the running time of Karp's algorithm although it does not change the running time asymptotically.

The Karp2 algorithm is a space efficient version of Karp's algorithm[1]. Karp's algorithm takes up $\Theta(n^2)$ space in order to store the D-values. The Karp2 algorithm reduces this space requirement to $\Theta(n)$. The Karp2 algorithm performs two passes. In the first pass, it computes $D_n(v)$ for each node v without storing $D_k(v)$ for $k < n$. In the second pass, it computes the fraction in Karp's theorem as it computes each $D_k(v)$, $k < n$. The DG and HO algorithms also suffer from this large space complexity problem. Fortunately, the technique used in the Karp2 algorithm is also applicable to them.

2.4 Parametric shortest path algorithms

The KO algorithm [16] and the YTO algorithm [25] are in this category. The YTO algorithm is essentially an efficient implementation of the KO algorithm. These algorithms are based on the observation that the minimum cycle mean λ^* is the largest λ such that G_λ does not have any negative cycles. Thus, these algorithms start with $\lambda = -\infty$ and always maintain a tree of shortest paths to a source node s. These algorithms change λ incrementally so that the shortest path tree changes by one arc in each iteration. When a cycle of weight zero is detected in G_λ, that cycle is the cycle with the minimum mean.

2.5 Lawler's algorithm

Lawler's algorithm [17] is based on the same observation as the parametric shortest path algorithms. It also uses the fact that λ^* of G lies between the minimum and the maximum arc weights in G. Lawler's algorithm does a binary search over the possible values of λ^* and checks for a negative cycle in G_λ every iteration. If one is found, then the chosen λ is too large so it is decreased; if not, it is too small so it is increased. Lawler's algorithm terminates when the interval for the possible values of λ^* becomes too small. The size of that interval, ϵ, determines the precision of the algorithm.

[1] Suggested by S. Gaubert of INRIA, France.

Input: A strongly connected digraph $G = (V, E)$.
Output: The minimum cycle mean λ^* of G.

for each node $u \in V$ **do** $d(u) \leftarrow +\infty$
for each arc $(u, v) \in E$ **do**
  if $(w(u,v) < d(u))$ **then**
    $d(u) \leftarrow w(u,v)$; $\pi(u) \leftarrow v$ /* policy */
while (true) **do** /* Main loop - Iterate */
  /* Find the set E_π of policy arcs */
  $E_\pi \leftarrow \{(u, \pi(u)) \in E\}$
  /* Compute λ in the policy graph G_π */
  Examine every cycle in $G_\pi = (V, E_\pi)$.
  Let C be the cycle with the smallest mean in G_π.
  $\lambda \leftarrow w(C)/|C|$
  Select an arbitrary node $u \in C$.
  /* Compute the node distances using the reverse BFS */
  if (there is a path from v to u in G_π) **then**
    $d(v) \leftarrow d(\pi(v)) + w(v, \pi(v)) - \lambda$
  /* Improve the node distances */
  $improved \leftarrow false$
  for each arc $(u, v) \in E$ **do**
    $\delta(u) \leftarrow d(u) - (d(v) + w(u,v) - \lambda)$
    if $(\delta(u) > 0)$ **then**
      if $(\delta(u) > \epsilon)$ **then** $improved \leftarrow true$
      $d(u) \leftarrow d(v) + w(u,v) - \lambda$; $\pi(u) \leftarrow v$
  /* If not much improvement in the node distances, exit */
  if ($NOT\ improved$) **then return** λ

Figure 1: Howard's algorithm.

2.6 Howard's Algorithm

Howard's algorithm [6] is given in Figure 1. It is similar to the style of the parametric shortest path algorithms except that it starts with a large λ and decreases λ until the shortest paths in G_λ are well defined. It computes λ on the *policy graph* which is simply a subgraph of G such that the out-degree of each node is exactly one. Note that the policy graph has n arcs. For a given λ, the algorithm attempts to find the shortest paths from every node to an chosen node u. In doing so, it either discovers that the shortest paths are well defined in which case the correct λ has been found or it discovers a negative cycle in G_λ. In the latter case, the negative cycle has a smaller mean weight than the current λ. In this case, λ can be updated to the mean weight of the new cycle and the process continues.

The beauty of Howard's algorithm is that each iteration of the algorithm is extremely simple and requires only $\Theta(m)$ time. Meanwhile, although it ensures that the value of λ is non-increasing from one iteration to another, it usually manages to make significant progress in decreasing its value in a very few number of iterations.

2.7 Scaling algorithms

The OA1 and OA2 algorithms [21] are in this category. They assume that the arc weights are integers bounded by W. If W is polynomial in n, then these algorithms are asymptotically the fastest algorithms. The OA2 algorithm applies scaling to a hybrid version of an assignment algorithm, called the auction algorithm, and the successive shortest path algorithm. It uses an approximate binary search technique. The OA1 algorithm is the same as the OA2 algorithm except that it does not use the successive shortest path algorithm.

3 Experimental Framework

We programmed the algorithms in C++ using the LEDA library version 3.4.1. This library is a template library for efficient data types and algorithms [20]. In order to ensure uniformity of implementation, all the algorithms were implemented in the same style by one of us. We also flattened each algorithm in that we manually inlined all the functions other than the functions needed by the LEDA data types. This eliminated the overhead of function invocations. The total size of the programs is approximately 2700 lines of C++ code.

We compiled and linked each program using the Sun C++ compiler CC version 3.0.1 under the O4 optimization option. We conducted the experiments on a Sun Sparc 20 Model 512 with two CPUs, 64 MB of main memory, and 105 MB of swap space. The operating system was SunOS version 5.5.1.

We did two sets of experiments: one to measure the running time of each algorithm and another to count the key operations of each algorithm, as advocated in [2]. Our test suite contained random graphs, generated using SPRAND [5], and cyclic sequential multi-level logic benchmark circuits, obtained from the 1991 Logic Synthesis and Optimization Benchmarks [24]. SPRAND produces a graph with n nodes and m arcs by first building a Hamiltonian cycle on the nodes and then adding $m - n$ arcs at random. This cycle makes the graph strongly connected. We generated 10 versions of each random graph. The experimental data reported for these graphs in this paper are the average over these 10 versions. The arc weights in the random graphs were uniformly distributed in [1, 10000], which is the default weight interval in SPRAND. Due to lack of space, we do not present the experimental results for the benchmark circuits: they can be found in [9].

The properties of the random graphs in our test suite are given together with the running times in Tables 2. We used sparse random graphs in our test suite because real circuits are sparse and we wanted our random graphs to represent them as closely as possible. We did more experiments than were reported in this paper. Since the trend for the dependence of the performance on the graph parameters is evident from those that we included in this paper, we did not see any need to include more experimental results. When doing our experiments, we tried to follow the guidelines in [1]. When comparing the algorithms using their operation counts, we compared only the relevant ones because all the algorithms do not have the same kind of operations. For instance, we compared only the KO and YTO algorithms for the number of heap operations.

4 Experimental Results and Observations

4.1 The minimum cycle mean and the graph parameters

For the random graphs, the minimum cycle mean is almost independent of the number of nodes, and it changes inversely with the density of the graph. This observation is expected because as the density of a graph increases, the graph contains more cycles and the critical cycles get smaller. This simple observation will be used to explain the behavior of some of the algorithms.

4.2 KO versus YTO

In our implementation, both algorithms use Fibonacci heaps, which is the default heap data structure in LEDA. Since the YTO algorithm is essentially an implementation of the KO algorithm using Fibonacci heaps, their use in the YTO algorithm was a natural choice. Their use in the KO algorithm was preferred to make these two algorithms comparable.

From the experimental results reported in [9], we can see that both the algorithms perform almost the same number of iterations on each test case; however, the YTO algorithm provides savings in the number of heap operations, especially in the number of insertions. The savings are more on the random graphs, and they get better as the density increases because the rate of increase in these numbers in the KO algorithm is larger. Their running times are comparable but the YTO algorithm performs a bit faster when the density increases. This is expected because the YTO algorithm performs fewer heap operations.

4.3 Number of iterations

Burns', KO, YTO, and Howard's algorithms perform a number of iterations before they converge. An upper bound on the number of iterations for the first three algorithms is n^2, and that for Howard's algorithm is the product of the out-degrees of all the nodes. We have also measured the value of k when the HO algorithm terminates. We refer to this value as "the number of iterations" of the HO algorithm although it is not one in the sense of the other algorithms. It is always less than n.

From the experimental results reported in [9], the number of iterations is always less than the number of nodes for each algorithm although Howard's algorithm creates a few anomalies. It seems that unless $n = m$, the number of iterations for the first three algorithms is around $n/2$ on the random graphs, each of which is strongly connected. Moreover, Burns' algorithm performs fewer number of iterations than the KO algorithm, and the KO and YTO algorithms perform the same number of iterations. The number of iterations of the Howard's algorithm is drastically small compared to the other algorithms. In [6], it is conjectured that the number of iterations is $O(\lg n)$ on the average, and it is $O(m)$ in the worst case. Our experiments support the worst case conjecture. They also show that the number of iterations for Howard's algorithm and the HO algorithm gets smaller as the density of the graph increases although some anomalies exist. This can be explained by the first observation.

4.4 Karp's algorithm and its variants

From the experimental results reported in [9], it seems that the improvement achieved by the DG algorithm in the number of arcs visited during the computation of $D_k(v)$ for each v is very small on the random graphs, indicating that it is not effective for dense graphs. The improvement on the circuits is far better, which explains the better performance of the DG algorithm over Karp's algorithm.

The space efficient version of Karp's algorithm, the Karp2 algorithm, roughly doubles its running time, as expected. The space efficiency of the Karp2 algorithm is directly applicable to the DG and HO algorithms. The most effective improvement on the Karp's algorithm is the HO algorithm. Its running time is even better than those algorithms which are asymptotically faster than it. Extrapolating from the Karp2 algorithm, we can say that the space efficient version of the HO algorithm will double its running time, which still maintains its superiority to most of the other algorithms.

4.5 Running times

The running time comparisons are given in Tables 2. The results show the following: Howard's algorithm is the fastest by a great margin. The HO algorithm ranks second, which indicates that the early termination scheme in the HO algorithm is very effective. The slowest algorithm is Lawler's algorithm.

The good performance of Karp's algorithm, especially on small test cases, is mostly due to its simplicity; it contains three simple nested loops. Its simplicity facilitates its optimization by a compiler, e.g., when compiled without optimization, the DG algorithm almost always beats it. However, as the number of nodes gets larger, its performance degrades more rapidly.

Burns' algorithm is slower than the KO and YTO algorithms although it performs fewer number of iterations and it does not perform expensive operations such as heap operations. We attribute this behavior to the fact that it is not incremental; every iteration builds from the scratch.

The OA1 and OA2 algorithms are not as fast as their running time implies. They are in general slower than Karp's algorithm. We attribute much of this to their complexity; they are more difficult to optimize than the other algorithms.

5 Conclusions and Future Work

We have presented efficient algorithms for the minimum mean cycle problem. This paper is the first study that brings them to the attention of the CAD community. We have systematically compared these algorithms on random graphs as well as benchmark circuits and provide important insights into their individual performance as well as relative performance in practice. One of the most surprising results of this study is that Howard's algorithm is by far the fastest algorithm on the graphs tested in this study. Unfortunately, the known bounds on the running time of this algorithm, including our bounds in [9], are exponential. We are working on improving these algorithms based on the insight that we have obtained from this study. So far, we have improved Howard's algorithm and Lawler's algorithm.

Acknowledgments

The authors would like to acknowledge support from the following awards: NSF MIP 95-01615 (CAREER), NSF CCR-9806898, NSF CCR-9625844, DARPA DABT63-98-C-0045, University of California MICRO program, the Interstate Electronics Fellowship, and the DAC Design Automation Graduate Scholarship.

References

[1] Ahuja, R. K., Kodialam, M., Mishra, A. K., and Orlin, J. B. Computational investigation of maximum flow algorithms. *European J. of Operational Research*, 97 (1997), 509–542.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network Flows.* Prentice Hall, Upper Saddle River, NJ, USA, 1993.

[3] Bacelli, F., Cohen, G., Olsder, G. J., and Quadrat, J.-P. *Synchronization and Linearity.* John Wiley & Sons, New York, NY, USA, 1992.

Table 2: The running time comparisons of Burns', KO, YTO, Howard's, HO, Karp's, DG, Lawler's, Karp2, and OA1 algorithms on the random graphs. For the cases marked with N/A, we could not get a result in many hours.

n	m	Burns	KO	YTO	Howard	HO	Karp	DG	Lawler	Karp2	OA1
512	512	3.48	1.51	1.67	0.01	1.00	0.79	0.06	11.09	1.41	328.88
512	768	2.34	1.04	1.12	0.16	0.32	0.98	1.03	6.51	1.83	5.80
512	1024	2.72	1.21	1.21	6.75	0.29	1.17	1.26	9.26	2.25	5.66
512	1280	4.11	1.82	1.73	0.17	0.31	1.37	1.47	10.62	2.71	6.98
512	1536	3.52	1.59	1.52	0.13	0.27	1.57	1.69	10.98	2.87	6.51
1024	1024	13.98	5.87	6.50	0.02	4.03	3.36	0.25	44.82	6.72	2790.12
1024	1536	10.17	4.41	4.61	0.34	1.07	4.17	4.66	34.67	7.87	12.34
1024	2048	11.32	4.98	4.99	0.21	0.84	5.05	5.64	30.33	9.04	13.78
1024	2560	15.16	6.74	6.62	0.23	0.94	5.91	6.63	54.77	10.82	23.67
1024	3072	13.91	6.25	5.90	0.22	0.87	6.77	7.56	51.91	14.60	17.13
2048	2048	55.88	23.13	25.46	0.04	16.45	13.48	1.02	186.35	21.80	20110.28
2048	3072	44.55	20.37	22.19	0.64	4.26	17.14	19.45	178.86	29.65	62.81
2048	4096	42.88	20.59	20.31	0.88	3.14	21.87	24.96	165.61	42.25	37.04
2048	5120	63.22	30.95	29.95	0.76	3.56	27.10	30.83	221.90	53.30	80.97
2048	6144	73.92	36.56	34.61	0.80	3.53	32.86	37.05	244.05	64.89	85.87
4096	4096	218.31	91.50	100.40	0.07	N/A	55.76	4.56	659.74	89.59	N/A
4096	6144	161.07	79.09	81.05	7.00	N/A	76.82	86.64	736.16	135.46	N/A
4096	8192	167.63	88.86	88.01	1.47	N/A	103.13	115.81	781.84	195.35	N/A
4096	10240	242.75	132.26	130.01	1.62	N/A	129.03	144.75	1305.47	259.19	N/A
4096	12288	236.71	139.22	137.87	13.84	N/A	156.70	173.94	1132.57	313.06	N/A
8192	8192	826.08	363.45	398.11	0.14	N/A	N/A	N/A	2819.30	355.00	N/A
8192	12288	559.88	306.52	329.73	4.09	N/A	N/A	N/A	2949.25	595.02	N/A
8192	16384	626.65	382.82	380.58	4.53	N/A	N/A	N/A	3708.98	858.01	N/A
8192	20480	840.50	536.50	524.82	4.73	N/A	N/A	N/A	5112.67	1110.84	N/A
8192	24576	874.21	609.60	587.51	5.57	N/A	N/A	N/A	5417.58	1905.20	N/A

[4] Burns, S. M. Performance analysis and optimization of asynchronous circuits. PhD thesis, California Institute of Technology, 1991.

[5] Cherkassky, B. V., Goldberg, A. V., and Radzik, T. Shortest path algorithms: Theory and experimental evaluation. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms* (1994), pp. 516–525.

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

[7] Cuninghame-Green, R. A., and Yixun, L. Maximum cycle-means of weighted digraphs. *Applied Math.-JCU 11* (1996), 225–34.

[8] Dasdan, A., and Gupta, R. K. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Trans. Computer-Aided Design 17*, 10 (Oct. 1998).

[9] Dasdan, A., Irani, S., and Gupta, R. K. An experimental study of minimum mean cycle algorithms. Tech. rep., University of California, Irvine, July 1998.

[10] Gerez, S. H., de Groot, S. M. H., and Herrmann, O. E. A polynomial-time algorithm for the computation of the iteration-period bound in recursive data-flow graphs. *IEEE Trans. on Circuits and Syst.-1 39*, 1 (Jan. 1992), 49–52.

[11] Gondran, M., and Minoux, M. *Graphs and Algorithms*. John Wiley and Sons, New York, NY, USA, 1984.

[12] Hartmann, M., and Orlin, J. B. Finding minimum cost to time ratio cycles with small integral transit times. *Networks 23* (1993), 567–74.

[13] Hulgaard, H., Burns, S. M., Amon, T., and Borriello, G. An algorithm for exact bounds on the time seperation of events in concurrent systems. *IEEE Trans. Comput. 44*, 11 (Nov. 1995), 1306–17.

[14] Ito, K., and Parhi, K. K. Determining the minimum iteration period of an algorithm. *J. VLSI Signal Processing 11*, 3 (Dec. 1995), 229–44.

[15] Karp, R. M. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics 23* (1978), 309–11.

[16] Karp, R. M., and Orlin, J. B. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics 3* (1981), 37–45.

[17] Lawler, E. L. *Combinatorial Optimization: Networks and Matroids*. Holt, Reinhart, and Winston, New York, NY, USA, 1976.

[18] Mathur, A., Dasdan, A., and Gupta, R. K. Rate analysis of embedded systems. *ACM Trans. on Design Automation of Electronic Systems 3*, 3 (July 1998).

[19] Megiddo, N. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research 4*, 4 (Nov. 1979), 414–424.

[20] Mehlhorn, K., and Naher, S. LEDA: A platform for combinatorial and geometric computing. *Comm. of the ACM 38*, 1 (1995), 96–102.

[21] Orlin, J. B., and Ahuja, R. K. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming 54* (1992), 41–56.

[22] Szymanski, T. G. Computing optimal clock schedules. In *Proc. 29th Design Automation Conf.* (1992), ACM/IEEE, pp. 399–404.

[23] Teich, J., Sriram, S., Thiele, L., and Martin, M. Performance analysis and optimization of mixed asynchronous synchronous systems. *IEEE Trans. Computer-Aided Design 16*, 5 (May 1997), 473–84.

[24] Yang, S. Logic synthesis and optimization benchmarks user guide version 3.0. Tech. rep., Microelectronics Center of North Carolina, Jan. 1991.

[25] Young, N. E., Tarjan, R. E., and Orlin, J. B. Faster parametric shortest path and minimum-balance algorithms. *Networks 21* (1991), 205–21.