Contraction versus Relaxation: A Comparison of Two Approaches for the Negative Cost Cycle Detection Problem

K. Subramani and L. Kovalchick

LCSEE,
West Virginia University,
Morgantown, WV
{ksmani,lynn}@csee.wvu.edu

Abstract. In this paper, we develop a greedy algorithm for the negative cost cycle detection problem and empirically contrast its performance with the "standard" Bellman-Ford (BF) algorithm for the same problem. Our experiments indicate that the greedy approach is superior to the dynamic programming approach of BF, on a wide variety of inputs.

1 Introduction

In this paper, we are concerned with the Negative Cost Cycle Detection problem (NEG): *Given a directed graph* $\mathbf{G} = <\mathbf{V}, \mathbf{E}>$*, where* $|\mathbf{V}| = n$ *and* $|\mathbf{E}| = m$*, and a cost function* $c : \mathbf{E} \rightarrow \Re$*, is there a negative cost cycle in* $\mathbf{G}$*?*

Our main contribution is the proposal of a greedy algorithm for NEG, based on vertex contraction. *All approaches to the negative cost cycle problem in the literature are based on dynamic programming; our approach is the first and only greedy approach to this problem, that we know of.* Scaling approaches have also been proposed for NEG ([Gol95]); however, these algorithms are efficient, only when the edge-weights are small integers. We do not place any restrictions on the edge costs. We note that the problem, as specified, is a decision problem, in that all that is asked of an algorithm is to *detect* the presence of a negative cycle. This problem finds application in a wide variety of areas such as Constraint Analysis [DMP91], Compiler Construction [Pug92], VLSI Design [WE94] and Scheduling [Sub02].

Our experiments indicate that Vertex Contraction is an effective alternative to the "standard" Bellman-Ford (BF) algorithm for the same problem; this is most surprising since in the case of sparse graphs, BF is provably superior to Vertex Contraction (from the perspective of asymptotic analysis).

2 The Vertex-Contraction Algorithm

The *vertex contraction* procedure consists of eliminating a vertex from the input graph, by merging all its incoming and outgoing edges. Consider a vertex v_i

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 377–387, 2003.

with incoming edge e_{ki} and outgoing edge e_{ij}. When v_i is contracted, e_{ki} and e_{ij} are deleted and a single edge e'_{kj} is added with cost $c_{ki} + c_{ij}$. This process is repeated for each pair of incoming and outgoing edges. Consider the edge e'_{kj} that is created by the contraction; it falls into one of the following categories:

1. It is the first edge between vertex v_k and v_j. In this case, nothing more is to be done.
2. An edge e_{kj} already existed between v_k and v_j, prior to the contraction of v_i. In this case, if $c'_{kj} < c_{kj}$, keep the new edge and delete the previously existing edge (since it is redundant); otherwise delete the new edge (since it is redundant).

Algorithm (2.1) is a formal description of our technique.

```
Function NEGATIVE-COST-CYCLE($\mathbf{G}, n$)
for ($i = 1$ to $n$) do
    VERTEX-CONTRACT($\mathbf{G}, v_i$)
end for
return(false)
```

Algorithm 2.1: Negative cost cycle detection

We defer a formal proof of the correctness of the VC algorithm to the journal version of this paper. In the full version, an analysis of VC is also provided; we show that the algorithm runs in worst case time $O(n^3)$.

Thus, for dense graphs, Algorithm (2.1) is competitive with Bellman-Ford (BF); however for sparse graphs, the situation is not so sanguine. For instance, an adversary could provide the graph in Figure (1) as input.

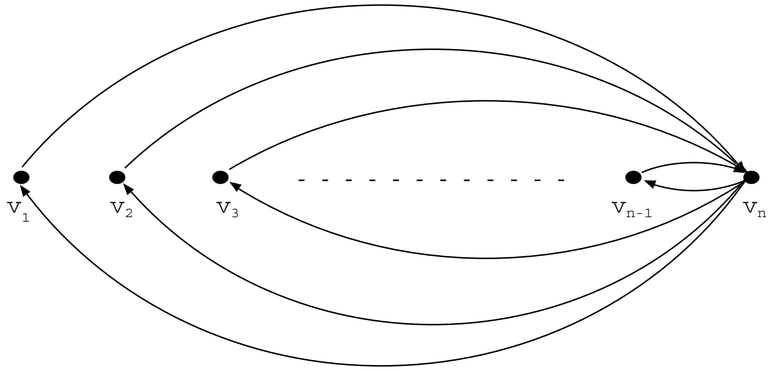

Fig. 1. Sparse graph that becomes dense after vertex contraction

```
Function VERTEX-CONTRACT($\mathbf{G}, v_i$)
for ($k = 1$ to $n$) do
  for ($j = 1$ to $n$) do
    if ($e_{ki}$ and $e_{ij}$ exist) then
      {Let $c_{kj}$ denote the cost of the existing edge between $v_k$ and $v_j$; note that $c_{kj} = \infty$ if there does not exist such an edge}
      Create edge $e'_{kj}$ with cost $c'_{kj} = c_{ki} + c_{ij}$
      Delete edges $e_{ki}$ and $e_{ij}$ from $\mathbf{G}$
      if ($j = k$) then
        {A cycle has been detected}
        if ($c'_{jj} < 0$) then
          return(true)
        else
          Delete edge $e_{jj}$
        end if
      else
        if ($c'_{kj} < c_{kj}$) then
          Replace existing edge $e_{kj}$ with $e'_{kj}$ in $\mathbf{G}$
        else
          Delete edge $e'_{kj}$
        end if
      end if
    end if
  end for
end for
```

Algorithm 2.2: Vertex Contraction

The above graph is sparse and has exactly $2 \cdot (n-1)$ edges. Observe that if vertex v_n is contracted first, the resultant graph is the complete graph on $n-1$ vertices and therefore dense. We call this graph *the cruel adversary*; in our experiments, we made it a point to contrast the performance of the vertex contraction algorithm with BF on this input. It is clear that any well-defined order of selecting the next vertex to be contracted is susceptible to attack by a malicious adversary; we could however choose the vertex to be contracted at random, without affecting the correctness of the algorithm. We have implemented Algorithm (2.1) in two different ways; in one implementation, the vertex to be contracted is chosen in a well-defined order, whereas in the second implementation, it is chosen at random. Algorithm (2.3) is a formal description of the random vertex contraction algorithm.

3 Implementation

Our experiments are classified into various categories, based on the following criteria:

1. Type of input graph - Sparse with many small negative cycles (Type A), Sparse with a few long negative cycles (Type B), Dense with many small

```
Function RANDOM-NEGATIVE-COST-CYCLE($\mathbf{G}, n$)
Generate a random permutation $\Pi$ of the set $\{1, 2, 3 \ldots, n\}$.
for ($i = 1$ to $n$) do
    VERTEX-CONTRACT($\mathbf{G}, v_{\Pi(i)}$)
end for
return(false)
```

Algorithm 2.3: Random negative cost cycle detection algorithm

negative cycles (Type C), Dense with a few long negative cycles (Type D), and the Cruel Adversary (Type E).

2. Type of Algorithm - Bellman-Ford (BF), Vertex-Contraction (VC) or Random Vertex-Contraction (RVC).
3. Type of Graph Data Structure - Simple Pointer or Array of Pointers.

All times recorded were averaged over 5 executions of each implementation.

3.1 Machine Characteristics

Table 1. Implementation System.

Machine Model	Silicon Graphics Onyx2
Processors	IR2/R10 250 Mhz
Cache	8 MB
Memory	2 GB
Operating System	IRIX 6.5.15
Language	C
Software	gcc

3.2 Graph Data Structures

Two different types of graph data structures were used for the experiments. We implemented BF, VC and RVC with an array of pointers structure and a simple pointer structure.

The array of pointers structure is a new representation. This representation makes use of an array of n pointers, one for each of the n vertices of the graph. Each pointer points to an n element array, which corresponds to the n vertices of the graph. Initially all entries of the array are assigned an undefined value. For a vertex v_i, if there exists an edge from v_i to another vertex v_j, position v_j of the array that v_i points to is assigned the cost of the edge between v_i and v_j. It should be noted that this representation is different from the adjacency-matrix representation [CLR92].

The simple pointer structure, also known as the adjacency-list representation [CLR92], requires only linear space. This representation makes use of an array of n lists, one for each of the n vertices of the graph.

3.3 Experimental Setup for Sparse Graphs

Sparse graphs were generated using the generator developed by Andrew Goldberg [CG96], which generates multiple edges between two vertices. Sparse graphs are defined as graphs with $o(n \cdot \log n)$ edges. We generated each graph 5 times using 5 different seeds for the random number generator.

Graphs of Type A and B were tested, with a number of vertices ranging from 500 to 5,500 in increments of 500.

We define a small negative cycle as one consisting of at most $\frac{n}{100}$ vertices. We define a long negative cycle as one consisting of $\Omega(\frac{n}{2})$ vertices. The number of long negative cycles in the input graphs was set to 4.

n	Array of Pointers (Time in Seconds)	
	VC	BF
500	0.15351	2.80657
750	0.50453	9.37442
1,000	1.58202	27.9044
1,250	2.23023	54.0744
1,500	4.74535	105.143
1,750	5.55235	156.953
2,000	12.7588	257.852
2,250	19.5588	337.136
2,500	13.9183	514.046
2,750	24.3229	624.652
3,000	30.4645	883.024
3,250	34.8372	1034.04
3,500	49.6497	1400.41
3,750	48.4852	1606.85
4,000	88.8478	2104.50
4,250	70.4305	2319.88
4,500	132.506	3094.30
4,750	82.0854	3180.42
5,000	108.178	4229.53
5,250	116.699	4377.80
5,500	133.606	5453.65

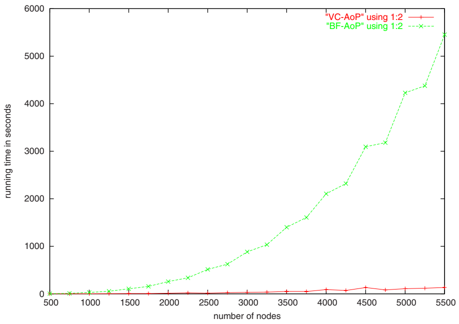

Fig. 2. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Array of Pointer (AoP) implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type A graphs.

n	Simple Pointer (Time in Seconds)	
	VC	BF
500	0.003933	1.65399
750	0.007623	5.19749
1,000	0.009573	11.8637
1,250	0.023780	22.5836
1,500	0.013797	38.9001
1,750	0.013525	64.8949
2,000	0.022071	103.797
2,250	0.022178	155.955
2,500	0.025375	222.823
2,750	0.030861	304.137
3,000	0.040336	403.182
3,250	0.046731	519.489
3,500	0.047264	656.995
3,750	0.071233	814.206
4,000	0.063790	995.579
4,250	0.073681	1199.38
4,500	0.101693	1433.95
4,750	0.083590	1688.46
5,000	0.124874	1981.08
5,250	0.084357	2295.98
5,500	0.087477	2650.91

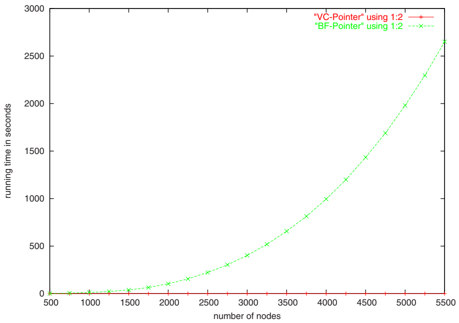

Fig. 3. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Simple Pointer implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type A graphs.

n	Array of Pointers (Time in Seconds)	
	VC	BF
500	0.22578	2.80053
750	0.51732	9.40496
1,000	1.63598	25.2701
1,250	2.76916	52.7881
1,500	4.03085	103.765
1,750	4.58197	152.435
2,000	11.8345	253.484
2,250	20.4233	330.726
2,500	13.9014	502.027
2,750	23.9882	607.284
3,000	27.1102	875.921
3,250	35.6303	995.875
3,500	49.7201	1383.19
3,750	48.6051	1577.46
4,000	77.5183	2071.36
4,250	65.2763	2307.49
4,500	120.153	2978.97
4,750	83.6087	3209.21
5,000	92.0987	4076.75
5,250	151.066	4376.84
5,500	130.703	5408.41

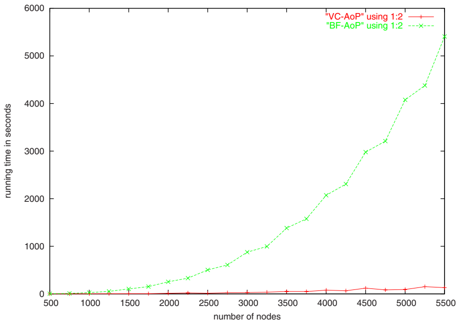

Fig. 4. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Array of Pointer (AoP) implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type B graphs.

	Simple Pointer (Time in Seconds)	
n	VC	BF
500	0.004119	1.65394
750	0.008052	5.20284
1,000	0.011301	11.8367
1,250	0.022105	22.5755
1,500	0.099097	38.8326
1,750	0.022232	64.7921
2,000	0.021255	103.191
2,250	0.037886	154.749
2,500	0.026206	222.234
2,750	0.030613	303.657
3,000	0.037332	403.146
3,250	0.050565	518.724
3,500	0.047130	655.168
3,750	0.078139	813.916
4,000	0.188993	993.696
4,250	0.078959	1199.66
4,500	0.106838	1432.46
4,750	0.059203	1690.80
5,000	0.128170	1977.29
5,250	0.096562	2293.64
5,500	0.114865	2646.47

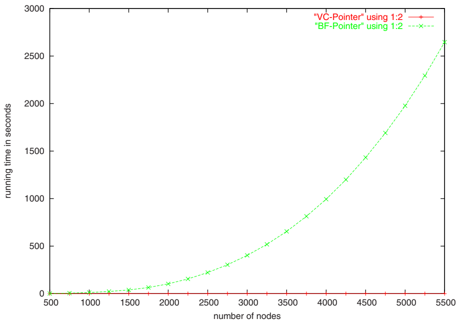

Fig. 5. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Simple Pointer implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type B graphs.

3.4 Conclusions

It is easy to see from the tables and graphs in Figure (2) through Figure (5) that VC outperforms BF using either data structure; this is true for both types of sparse graphs that were tested. We conclude that VC is far superior to BF for sparse graphs.

An asymptotic analysis would indicate that BF is superior to VC for dense graphs, although, our experiments contradict this indication.

3.5 Experimental Setup for Dense Graphs

Dense graphs were generated using the generator developed by Andrew Goldberg [CG96]. Dense graphs were defined as those with $\Omega(\frac{n^2}{8})$ edges. We generated each graph 5 times using 5 different seeds for the random number generator.

Graphs of Type C and D were tested, with a number of vertices ranging from 125 to 1,875 in increments of 125, with small negative cycles and long negative cycles defined as in Section §3.3.

n	Array of Pointers (Time in Seconds)	
	VC	BF
125	0.03830	0.07012
250	0.05997	0.52512
375	0.15247	1.74020
500	0.28095	4.08695
625	0.46288	7.98574
750	0.65885	13.7706
875	1.48780	21.9978
1,000	1.55311	34.5631
1,125	2.97252	51.5897
1,250	3.37425	74.6079
1,375	4.37236	100.182
1,500	7.30295	132.641
1,625	6.22661	168.864
1,750	8.63348	210.632
1,875	8.00939	260.838

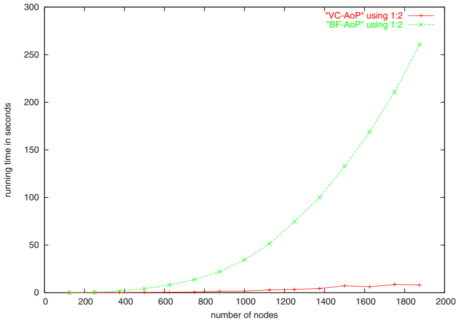

Fig. 6. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Array of Pointer (AoP) implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type C graphs.

n	Simple Pointer (Time in Seconds)	
	VC	BF
125	0.00048	0.09019
250	0.00194	1.02020
375	0.00303	4.50625
500	0.00675	13.2450
625	0.00750	30.9083
750	0.01562	62.0953
875	0.03498	123.824
1,000	0.09200	293.591
1,125	0.11334	672.256
1,250	0.19100	1350.14
1,375	0.25657	2447.76
1,500	0.42457	4079.35
1,625	0.41033	6346.27
1,750	0.65944	9445.69
1,875	0.99798	13480.9

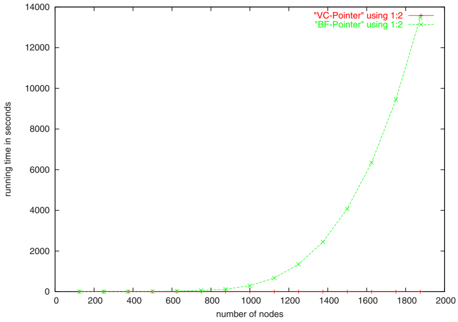

Fig. 7. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Simple Pointer implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type C graphs.

	Array of Pointers (Time in Seconds)	
n	VC	BF
125	0.00146	0.06872
250	0.06294	0.52747
375	0.21069	1.73264
500	0.30008	4.09149
625	0.65476	7.98092
750	0.74009	13.7807
875	2.03858	21.9476
1,000	1.50686	35.0864
1,125	3.64647	51.5824
1,250	3.17255	72.8195
1,375	6.92824	101.460
1,500	6.88772	133.293
1,625	6.67336	167.244
1,750	7.48221	212.698
1,875	15.9974	263.763

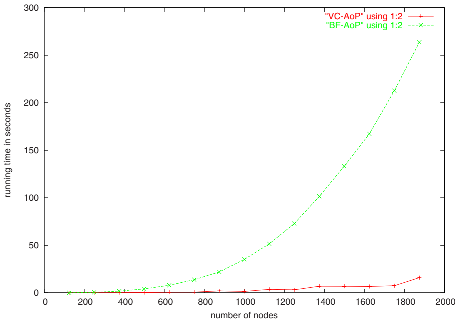

Fig. 8. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Array of Pointer (AoP) implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type D graphs.

	Simple Pointer (Time in Seconds)	
n	VC	BF
125	0.00069	0.08752
250	0.00185	1.02023
375	0.00488	4.44321
500	0.00706	13.2395
625	0.01379	30.6152
750	0.01765	62.0956
875	0.05033	122.713
1,000	0.07456	293.617
1,125	0.20491	664.932
1,250	0.22301	1348.96
1,375	0.41339	2426.95
1,500	0.39452	4079.71
1,625	1.00850	6299.06
1,750	0.59050	9447.84
1,875	1.43688	13418.7

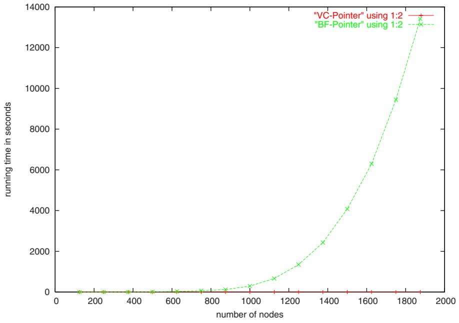

Fig. 9. Comparison of Vertex Contraction (VC), and Bellman-Ford (BF) Simple Pointer implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type D graphs.

3.6 Conclusions

It is easy to see from the tables and graphs in Figure (6) through Figure (9) that VC outperforms BF using either data structure; this is true with both types of dense graphs that were tested. We conclude that VC is far superior to BF for dense graphs.

3.7 Experimental Setup for Cruel Adversary Graphs

The cruel adversary is generated by specifying the number of vertices in the graph and the maximum cost for any edge.

For our experiments we generated graphs with vertices ranging from 125 to 1,875 in increments of 125.

	Array of Pointers (Time in Seconds)		
n	VC	RVC	BF
125	0.02168	0.05125	0.04738
250	0.16735	0.40114	0.34556
375	0.55377	1.35226	1.13898
500	1.29808	3.08334	2.66123
625	2.54713	6.22039	5.19088
750	4.38616	10.7309	9.02499
875	7.01832	15.6965	14.3931
1,000	10.9170	25.7463	23.6483
1,125	15.7832	37.1957	35.7989
1,250	21.9961	52.2092	54.5719
1,375	30.1900	71.4197	72.7322
1,500	41.3305	93.0058	94.8282
1,625	53.3580	66.8170	121.354
1,750	66.5882	147.186	152.360
1,875	83.2914	171.411	188.266

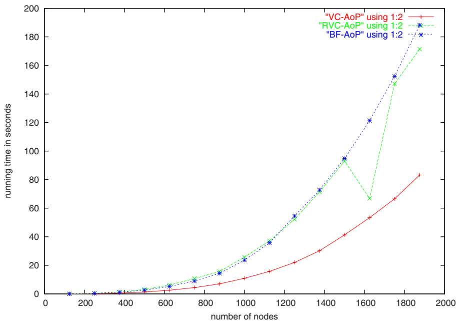

Fig. 10. Comparison of Vertex Contraction (VC), Random Vertex Contraction (RVC) and Bellman-Ford (BF) Array of Pointer (AoP) implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type E graphs.

	Simple Pointer (Time in Seconds)		
n	VC	RVC	BF
125	0.06657	0.01769	0.04572
250	0.48752	0.09771	0.33976
375	1.61851	0.47191	1.11752
500	4.55655	0.53125	2.61450
625	10.2743	2.45196	5.06600
750	19.7552	3.40885	8.70514
875	33.6204	5.76912	13.7683
1,000	52.5211	7.97386	20.4959
1,125	76.4936	25.6282	29.0852
1,250	106.248	33.1245	39.9211
1,375	144.869	9.84373	53.0680
1,500	187.058	67.5207	69.8548
1,625	244.296	66.2070	91.4537
1,750	304.338	78.6134	118.606
1,875	379.341	29.4672	151.796

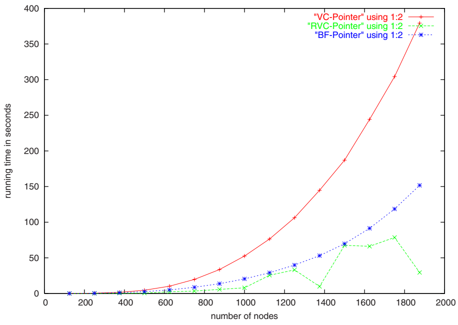

Fig. 11. Comparison of Vertex Contraction (VC), Random Vertex Contraction (RVC) and Bellman-Ford (BF) Simple Pointer implementation execution times (seconds) required to solve the Negative Cost Cycle problem for Type E graphs.

3.8 Conclusions

VC does considerably better than both RVC and BF, as observed from the table and graph in Figure (10) of the Array of Pointer implementation on Type E graphs. The results of RVC and BF are similar with RVC doing better in most instances.

VC does very poorly, as observed from the table and graph in Figure (11) of the Pointer implementation on Type E graphs. RVC does much better than VC and outperforms BF by a large margin on most instances. One conclusion that can be drawn from the data is that the time taken by RVC varies greatly depending on the random sequence of vertices chosen.

4 Conclusion

In this paper, we designed and analyzed a greedy algorithm called the vertex contraction algorithm (VC) for the negative cost cycle detection problem. Although vertex contraction is asymptotically inferior to the Bellman-Ford algorithm on sparse graphs, it is vastly superior from an empirical perspective.

We are currently working on two extensions: (a) Comparing our strategy with the Goldberg approach, (b) Combining the main idea of our approach, with heuristics such as contracting the vertex with the smallest degree-product.

References

[CG96] Boris V. Cherkassky and Andrew V. Goldberg. Negative-cycle detection algorithms. In Josep Díaz and Maria Serna, editors, *Algorithms—ESA '96, Fourth Annual European Symposium*, volume 1136 of *Lecture Notes in Computer Science*, pages 349–363, Barcelona, Spain, 25–27 September 1996. Springer.

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, June 1995.

[Pug92] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. *Comm. of the ACM*, 35(8):102–114, August 1992.

[Sub02] K. Subramani. An analysis of zero-clairvoyant scheduling. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the* 8^{th} *International Conference on Tools and Algorithms for the construction of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, April 2002.

[WE94] Neil H. Weste and Kamran Eshragian. *Principles of CMOS VLSI Design.* Addison Wesley, 1994.