Global Journal of Mathematical Sciences: Theory and Practical.
ISSN 0974-3200 Volume 3, Number 4 (2011), pp. 307-315
© International Research Publication House
http://www.irphouse.com

A Faster Negative Cycle Detection Algorithm

Saranya C.R.[1]* and Shobhalatha G.[2]

[1]Department of Mathematics, Sri Sathya Sai Institute of Higher Learning, Anantapur Campus, Anantapur, India
[2]Department of Mathematics, Sri Krishnadevaraya University, Anantapur, India
**Corresponding Author E-mail: saranya.chenchu@gmail.com*

Abstract

In this paper, we propose an algorithm for the negative cycle detection problem. An algorithm for the negative cycle problem combines a shortest path algorithm and a cycle detection strategy. On a graph with n vertices and m edges, our algorithm runs in O(n^2) time which is a better time bound for the case where n is much lesser than m. We use a cycle detection strategy which is a slight modification and culmination of the time out and walk to root strategies. This algorithm does not maintain a queue or stack of distance labels as the existing algorithms.

Keywords: Digraph, Weighted graph, Vertex set, Edge set, source, shortest path, constraint graph and Negative cycle.

2000 MSC: 05C38, 05C85, 68R10

Introduction

The shortest path problem with real(positive or negative)weights is the problem of finding the shortest distance from a specified vertex to all the vertices in the graph. Negative edge weights arise in a natural way when we reduce other problems to shortest-path problems. Negative weights are not merely a mathematical curiosity, on the contrary, they significantly extend the applicability of the shortest-path problems as a model for solving other problems. This potential utility is our motivation to search for efficient algorithms to solve network problems that involve negative weights. The negative cycle problem is to find a negative length cycle in a network or to prove that there are none. The Negative Cycle Detection problem has numerous applications in scheduling, circuit production, constraint programming and image processing. For example, in some linear programming applications with constraints of the form $x_j - x_i \leq b_k$ called difference constraints, the problem has feasible solution if and only if the corresponding constraint graph has no negative cycle.

The previously known algorithms for the problem are based on the famous Bellman –Ford – Moore(BF) [1, 6, 10] algorithm whose time bound is O(nm), where n is the number of vertices and m is the number of edges. With the additional assumption that arc lengths are integers bounded below by $-N \leq -2$, the bound O($\sqrt{n}$ m log N) of Goldberg [8] improved the Bellman-Ford-Moore bound for very large N, where N is the absolute value of most negative arc length. The Goldberg - Radzik algorithm [7], an incremental graph algorithm of Pallottino [11], an algorithm of Tarjan[12] all perform well on some classes of shortest path problems. C-H. Wong and Y-C Tan [5] gave some heuristics that can be used to improve the runtime of a wide range of commonly used algorithms for the negative cycle detection problem significantly, such as Bellman – Ford - Tarjan algorithm, Goldberg - Radzik algorithm and Bellman-Ford-Moore algorithm with Predecessor Array. It runs in O(n^2m) worst-case time. An O(n)-pass algorithm, called robust Dijkstra (RD) with bucket implementation and heap implementation was proposed by Cherkassky B.V. et.al.[3] which performed better than Wong and Tan's methods on many of the classes of graphs.

Every labeling algorithm terminates after a certain number of labeling operations in the absence of negative cycles. If this number is exceeded, we can stop and declare that the network has a negative cycle. This strategy is called time out strategy. Our algorithm takes in to account the number of times a particular vertex becomes the scanning vertex. If this number exceeds 2 we declare that the graph has negative cycle. When the labeling operation is applied on an arc (u, v), the walk to root strategy follows the parent pointers from u until it reaches v or s. If we stop at v, then it declares the presence of negative cycle. Our algorithm uses this strategy and whenever it reaches v or s, it checks for the distance of v and s, we declare the presence of negative cycle of distance of s is less than zero or if v becomes the scanning vertex for the third time. At each iteration our algorithm has the information about the vertex with the minimum distance among all the scanned and labeled vertices.

Definitions and Algorithmic Preparations

Given a weighted directed graph G=(V,E), with V[G] the vertex set and E[G] the edge set,a weight function and a function w : E →R, mapping edges to real-valued weights. The shortest path problem is the problem of finding shortest distances from a specified vertex to all other vertices. The weight of the path p =$(v_0, v_1, \ldots, v_k)$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w\,(v_{i-1}, v_i)$$

The shortest path weight from a vertex u to v is defined by

$$\delta(u,v) = \begin{cases} \min(\mathrm{w(p)}) : \text{if there is a path from u to v.} \\ \infty \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path with weight w(p)=$\delta(u, v)$.

For a given graph G = (V, E), the shortest path is represented using π[v] which maintains the **predecessor** of vertex v. π[v] is either another vertex or NIL. **Predecessor subgraph**, $G_\pi = (V\pi, E\pi)$ induced by the π values gives the shortest path tree. Where $V\pi$ is the set of vertices with non-NIL predecessors, plus the source and $E\pi$ is the directed edge set induced by the π values for vertices in $V\pi$.

The process of relaxing an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating d[v] and π[v]. If d[v] is greater than d[u] + w(u,v), then this process sets d[v] to d[u]+w(u,v) and updates π[v] to u.

For every vertex v, the algorithm maintains the following (i) distance from the source to v denoted by d[v] (ii) distance of a particular vertex in the previous iteration denoted by pd[v] (iii) the parent or the predecessor of the vertex v denoted by π[v] and (iv) the number of times a particular vertex has been scanned denoted by ns[v].

In addition, the algorithm puts all the vertices that have been newly labeled in a particular iteration in the set 'Labeled-New' and all the vertices which were labeled in the previous iterations are included in the set 'Labeled-Old'.

In the next section we discuss the structure of the algorithm and give the proposed algorithm, the step by step analysis of the algorithm followed by proofs for correctness of the algorithm are done in section 3 and in the last section we compare the new algorithm with few well known algorithms.

O(n^2) Algorithm

The Algorithm works as follows:

1. It starts with initialization of d[v] and pd [v] to ∞, π[v] to Nil and ns[v] to 0.
2. The iteration procedure starts with d[s] set to 0, a variable source assigned as s and the two sets Labeled-Old and Labeled-New set to Empty.
3. The For loop runs $2|V(G)|$ times, each time the loop is executed it does the following It checks for ns[source]>2 or d[s]<0. If either of them is satisfied the algorithm terminates as it has found the negative cycle. Otherwise, it starts labeling the other vertices only if it's distance in the previous iteration or it's previous distance is greater than the newly evaluated distance. At the end of every iteration the vertex with the minimum of all the distances is made the source for the next iteration. Once the set Labeled-Old becomes empty at the end of any iteration the algorithm terminates as it has obtained the solution.

Algorithm

```
1.  START
2.  For each $v \in v[G]$
3.  do d[v] ← ∞
4.  pd[v] ← ∞
5.  $\pi$[v] ←NIL
6.  ns[v] ← 0
7.  End For
8.  d[s] ←0
9.  source← s
10. Labeled-New ← Empty
11. Labeled-Old← Empty
12. For i ←1 to $2|V(G)|$
13. do ns[source] ← ns[source]+1
14. min-d-old ← ∞
15. if ( ns[source] > 2 OR d[s] < 0)
16. then display “NEGATIVE CYCLE DETECTED”
17. Exit Loop and STOP
18. End if
19. For each v ∈ Adj[source]
20. do pd[v] ← d[v]
21. if d[v]> d[source] + w(source,v)
22. then d[v] ← d[source] + w(source,v)
23. $\pi$[v] ←source
24. End if
25. if pd[v] > d[v]
26. then Labeled-New ← Labeled-New ∪ {v}
27. Labeled-Old ← Labeled-Old−{v}
28. min-d-new ← d[v]
29. min-v-new ← v
    else min-d-new ← ∞
31. if(min-d-new < min-d-old)
32. then min-d-old ← d[v]
33. min-v-old ← v
34. End if
35. End For
36. if ( min-v-old = source )
37. then ns[source] ← ns[source] −1
38. i← i−1
39. End if
40. For each v ∈ Labeled-Old
41. do if ( min-d-old > d[v] )
42. then min-d-old← d[v]
43. min-v-old ← v
```

```
44. End if
45. End For
46. source ← min-v-old
47. Labeled-Old ← Labeled-Old ∪ Labeled-New ←
48. Labeled-New ← Empty
49. if ( Labeled-Old = Empty )
50. then display “SOLUTION FOUND”
51. Exit loop and STOP
52. End if
53. Labeled-Old ← Labeled-Old $-\{source\}$
54. End For
55. STOP
```

Analysis of the algorithm

Lines 2 -7 initializes the values d[v],pd[v],π[v] and ns[v] for every vertex v in V. This takes O(n) time, where n is the number of vertices in G. As the iteration starts, say in i^{th} iteration, the vertices labeled are put in the set Labeled-New and these vertices are mixed with those in Labeled-Old $i + 1^{th}$ iteration. There is total of 2n iterations given by the for loop in line 12, min-d-old(minimum distance in previous iteration) is a variable that is introduced in line 14 to hold the minimum distance from the source to the vertex v, that will be scanned in the next iteration. At the beginning of each iteration this variable is set to ∞ so that the minimum of the newly labeled vertices can be found, it will give the correct minimum distance at the end of the iteration.

The algorithm checks for the situations where either, the number of times a particular vertex is scanned exceeds 2 or the distance of the source becomes less than 0. When either of this is satisfied, a ‘negative cycle has been detected’ and the algorithm stops. The case where these conditions are not satisfied, then we proceed to the first inner ‘for loop’, for each vertex adjacent to the vertex being scanned we have one iteration. At the beginning of the iteration, before d[v] undergoes a change it is saved in pd[v], as the previous distance. The edge (source,v) is relaxed. After relaxation process, if there is any change in d[v], that is if it has lost some distance then it can become a scanning vertex again. If there is no change in d[v] after relaxation then the variable min-d-new (minimum distance in the current iteration) is set to $\infty,$ this ensures that the value of min-d-new of the previous iteration is not carried forward, and the variable is min-v-new (vertex at minimum distance from source in the current iteration) is not updated.

After the complete execution of the first inner for loop, the second inner for loop starts at the end of which the minimum of all the labeled vertices can be obtained and stored in min-d-old and the corresponding vertex is stored in min-v-old(vertex at a minimum distance from source in the previous iteration). Now, the next vertex to be scanned is given by min-v-old and it is assigned to the variable source. The newly scanned vertices are combined with those in the Labeled-Old set, this new set now becomes the Labeled-Old set for the next iteration. When all the vertices have obtained the correct label, which happens in the absence of negative cycle, the set

Labeled-Old becomes empty, when it is satisfied, the solution is reached and the loop is exited. The inner for loops put together run n-1 times, because those vertices which are in Labeled-New does not belong Labeled-Old, in the worst case the total number of times the inner loops run is (n-1) times. The outer for loop runs 2n number of times. Hence the algorithm runs in O(n^2) time.

Proof of Correctness

Lemma 3.1: Let G be a graph with source 's', after a finite number of labeling operations if d[s]<0 then G contains a negative cycle.

proof:

Let us observe that the parent of any vertex has a finite distance label and all vertices with finite distance label has parents except for the source s. The source s can have a parent if and only if d(s) <0. Suppose d[s] <0, we shall prove that the graph has a negative cycle reachable from s. Since there is a path from source to any vertex in G, thus there is a path from source s to π[s],say p, since π is the parent of s, the path p along with the edge(π[v],s) forms a cycle in G. If *l*is the length of the path from s to π[s], let us name the vertices on the path as $v_0, v_1, \ldots, v_l$ where v_0 is the source s and v_l is the vertex π[s]. The vertex *vl* became the parent of s after the relaxation of the edge (v_l , v_0), so d[s] is nothing but d[v_l]+ w(v_l , v_0).

Since d[s] < 0

$$d[v_l] + d[v_{l-1}] + w(v_{l-1}, v_l) < 0. \qquad (1)$$

now v_{l-1} is the parent of v_l , d[v_l] is d[v_{l-1}] + w(v_{l-1}, v_l), thus (1) becomes

$$d[v_{l-1}] + w(v_{l-1}, v_l) + w(v_l, v_0) < 0$$

continuing in this way, we get,

$$w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{l-1}, v_l) + w(v_l, v_0) < 0$$

that is,

$$\sum_{i=1}^{l+1} w\,(v_{i-1}, v_i) < 0 \text{ where } v_{l+1} = v_0$$

The weight of the cycle is negative. Hence the proof.

Lemma 3.2: Let G be a graph with source 's', for any v ≠s, if v becomes a scanning vertex more than twice, that is if ns[v] > 2 then G contains a negative cycle.

Proof: Let us assume that v_0 is the vertex which becomes the scanning vertex for the third time and assume it is not in a negative cycle. Let $v_0$be in a cycle. Let l be the length of the cycle, and $v_0, v_1, \ldots, v_l$ are the vertices in the cycle where v_0 = v_l , then the weight of the cycle is given by $\sum_{i=1}^{l} w\,(v_{i-1}, v_i)$. Assume that

$$\sum_{i=1}^{l} w\,(v_{i-1}, v_i) > 0$$

When the cycle is traversed for the second time starting from v0, when it reaches the vertex v_0, d[v_0] is compared with d[v_0]+ $\sum_{i=1}^{l} w\ (v_{i-1}, v_i)$. Since the weight of the cycle is positive, we have

$$d[v_0] < d[v_0] + \sum_{i=1}^{l} w\ (v_{i-1}, v_i)$$

Hence, d[v_0] does not undergo any change once it reaches the minimum, which is a contradiction to the fact that it becomes the scanning vertex for the third time and it is not in a negative cycle. So $v_0$is in a negative cycle. Hence the proof.

Theorem 3.3: Given a weighted graph G=(V,E), in 2n iterations the algorithm either detects a negative cycle or gives a shortest path.

Proof: From Lemma 3.2, it is clear that a vertex can be scanning vertex at most twice, and from Lemma 3.1 if source becomes a scanning vertex then there must be a negative cycle. Therefore each vertex can become a scanning vertex only twice, excluding the sink or destination. Hence in 2n iterations either one of the conditions stated in lemma 3.1 or lemma 3.2 gets satisfied in the presence of a negative cycle. In the absence of a negative cycle, the solution is obtained within the 2n iterations.

Examples

The algorithm was applied on the following problem from the book 'Introduction to Algorithms,by Cormen, T. H.,et.al.(2001)[4]. The first problem is finding shortest path in the graph shown in fig.1. This graph does not contain a negative cycle and the result obtained after implementing the new algorithm was found to be correct.

The second problem is finding shortest path in the graph shown in fig.2. This graph does contain a negative cycle and the algorithm could detect the negative cycle in 7 iterations of the outer For loop.

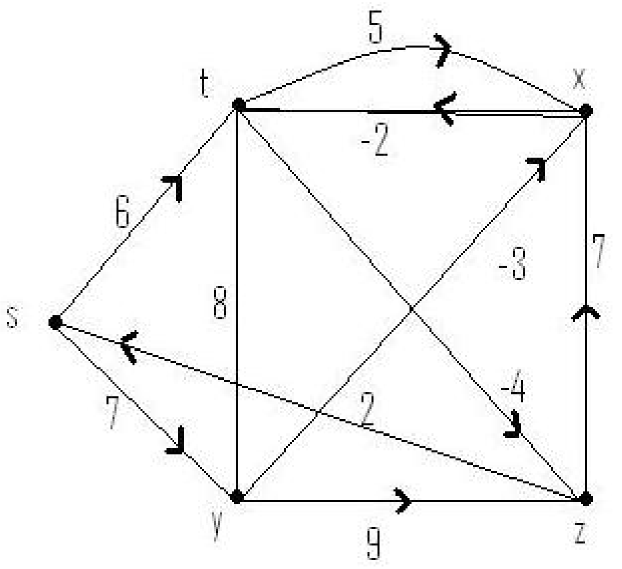

Figure 1: Network with Negative weights.

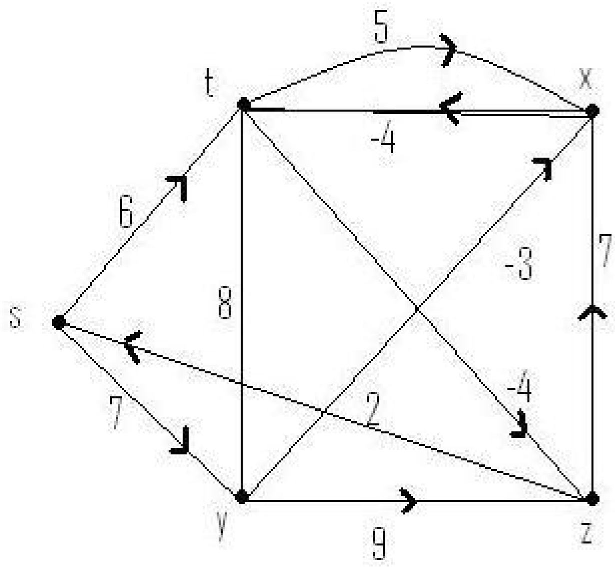

Figure 2: Network with Negative Cycle.

Comparison with few algorithms

In this section we compare the worst case time bounds of the few existing algorithms with the new algorithm. We have chosen few arbitrary values for n,m and N, where N is the absolute value of the most negative edge weight. The Bellman Ford Algorithm runs in O(nm) time, the algorithm by A.V.Goldberg runs in O($\sqrt{n}$ m log N) time. The algorithm given in this work has O(n^2) bound. The following table gives the comparison between the above mentioned algorithms, where we have found the worst case time for the algorithms, in practice these algorithm can even take lesser time than the evaluated values to produce the result.

Here BF represents the Bellman-Ford algorithm. Observe that for the networks where, m is greater than or equal to 2n, the new algorithm is always faster than Bellman-Ford algorithm. When compared with Goldberg algorithm, we see that, as given in the second case (*), the new algorithm is a better one only when the value of N is very large. But, for the cases were m is much larger than n, the new algorithm is faster than the other algorithms.

Table 1: Relative performance of BF, Goldberg and the new algorithm.

S. No.	n	m	N	BF(nm)	Goldberg ($\sqrt{n}$ m log N)	New Algorithm (n^2)
1	5	10	15	50	26.2982	25
2	50	150	*250	7500	2543.3995	2500
3	500	25000	4	12500000	336561.7668	250000
4	8193	24576	5	201351168	1554859726	134234112
5	16386	65537	3	1073889282	4002688528	536969220

Concluding Remarks

In this work we have introduced a negative cycle detection algorithm. We have shown by numerical examples that for reasonable sized graphs our algorithm outperforms the existing algorithms. We have compared our algorithm against the most efficient alternative like Goldberg's algorithm. The numerical examples suggest that the algorithm outperforms the other algorithms only under the conditions that either the number of edges 'm' is much larger than the number of vertices 'n' or the absolute value of most negative arc length 'N' is very large. This work can be extended to obtain a better algorithm which would overcome these limitations.

References

[1] Bellman, R. E., 1958, "On a routing problem", Quarterly of Applied Mathematics, 16, pp.87-90.

[2] Cherkassky, B. V., Goldberg, A. V., and Radzik, T., 1996, "Shortest Paths Algorithms: Theory and Experimental Evaluation", Math. Prog., 73, pp 129-174.

[3] Cherkassky, B. V., Loukas Georgiadis, Goldberg, A.V., Tarjan, R. E., Werneck, R. F., 2009, "Shortest Path Feasibility Algorithms: An Experimental Evaluation", J. Experimental Algorithmics (JEA), 14, section 2, article no.7.

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., 2001, Introduction to Algorithms, Second edition, MIT Press/McGraw-Hill.

[5] C-H. Wong and Y-C Tam, 2005, "Negative Cycle Detection Problem", Proc. 13th Annual European Symposium on Algorithms, pp. 652663.

[6] Ford, L.R. and Fulkerson, D.R., 1962, Flows in Networks, Princeton University,Princeton, NJ.

[7] Goldberg, A.V., Radzik, T., 1993, "A heuristic improvement of the Bellman-Ford algorithm", Applied Math. Lett. 6, pp 3 -6.

[8] Goldberg, A.V., 1995, "Scaling algorithms for the shortest paths problem", SIAM J. Comput. 24, pp 494 -504

[9] Goldfarb, D., Hao, J., Kai, S.-R., 1991, "Shortest path algorithms using dynamic breadth-first search", Networks 21, pp.29 -50.

[10] Moore, E.F.,1959," The shortest path through a maze", Proceedings of an International Symposium on the Theory of Switching, pp. 2-5 April 1957, Part II [The Annals of the Computation Laboratory of Harvard University Volume XXX] (H. Aiken, ed.), Harvard University Press, Cambridge, Massachusetts, pp. 285 -292.

[11] Pallottino, S., 1984, "Shortest-Path methods: complexity, interrelations and new propositions", Networks 14, pp.257 -267.

[12] Tarjan, R.E., 1981, Shortest Paths. Technical report, AT and T Bell Laboratories, Murray Hill, NJ