International Journal of Pure and Applied Mathematics

Volume 2 No. 3 2002, 323-333

ON A CLASS OF SHORTEST PATH ALGORITHMS WITH DISRUPTIVE SEARCH GRAPHS

Ming-Jer Tsai

Computer and Communications Research Laboratories
Industrial Technology Research Institute
R.M. 470, BLDG 51, No. 195, Sec. 4
Chung-Hsing R.D. Chutung
Hsinchu, Taiwan, ROC
e-mail: PAULIE@itri.org.tw

Abstract: A shortest path algorithm constructs the search graph when it searches a network. A disruptive search graph remains disconnected once it disconnects. In this paper, a class of shortest path algorithms that construct disruptive search graphs is found. By its aid, an arbitrary compromise between the time delay and the checking frequency for negative-cycle detection can be made.

AMS Subject Classification: 05C85, 68Q20, 68R10
Key Words: negative-cycle detection, search graph, shortest path algorithm

1. Introduction

Negative-cycle detection in a network is important for several applications such as minimal cost-to-time ratios [14], constraint generation on VLSI layouts [15], two-dimensional package placements [16] and min-cost flows [17], among others. There are two main works related to the negative-cycle detection problem. One detects negative cycles by executing shortest path algorithms [1], [7], [10], [11],

Received: April 8, 2002

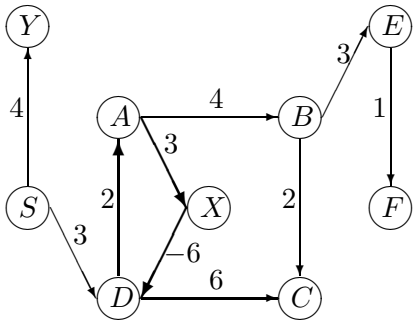

Figure 1: A negative-cycled network, where a negative cycle (D, A, X, D) forms. The number weighs the corresponding arc.

[15], [16], [18]. Another directly locates negative cycles without computing information about shortest paths [6], [8], [12], [13].

When a shortest path algorithm detects negative cycles in a network, there are two conflicting requirements: short time-delay and low checking-frequency. Many researchers limited the execution iterations and reported the presence of a negative cycle [1], [7], [11]. Other researchers made sufficient but not necessary criteria [10], [15], [18]. Still others check the relationship between the examined and scanned nodes [12], [16]. However, the first two methods have long time-delays, and the third has a high checking-frequency.

In this paper, a class of shortest path algorithms that construct disruptive search graphs is found. These algorithms are able to detect negative cycles by making arbitrary trade-offs between the above two conflicting requirements. The rest of this paper is organized as follows. Next section makes some notations. Also, there are formulas for search graphs in negative-cycled networks. Section three presents a class of shortest path algorithms with disruptive search graphs. Bertsekas's algorithm [2] and Dijkstra's algorithm [5] belong to the class after slightly modified. Finally, section four concludes this paper.

2. Preliminaries

Figure 1 shows a negative-cycled network. Definitions 1 and 2 define the search graph and the inverse search graph, respectively. Definition 3 defines the parent walk of a node in an inverse search graph. Examples of these definitions are

given in the next section.

Definition 1. A search graph $G^S(N^S, A^S)$ is a dynamic graph constructed when a shortest path algorithm searches a network $G(N, A)$ and is defined as follows.

1. $N^S = \emptyset$, $A^S = \emptyset$.
2. Whenever node u scans node v,
   (a) $N^S = N^S \cup v$,
   (b) $A^S = A^S - (w, v)$ if arc $(w, v) \in A^S$ and
   (c) $A^S = A^S \cup (u, v)$.

When a node u is examined, the node v is scanned if the label of node v decrements. And, it is noted that the search graph G^S is the shortest path tree [4], when the shortest path algorithm searches a non-negative-cycled network.

Definition 2. An inverse search graph $G^{IS}(N^{IS}, A^{IS})$ is defined as

1. If a node $v \in N^S$, node $v \in N^{IS}$.
2. If an arc $(u, v) \in A^S$, arc $(v, u) \in A^{IS}$.

Definition 3. The parent walk of node $u \in N^{IS}$, denoted by $PW(u)$, in inverse search graph G^{IS} is a walk that starts at a node u, then follows the arcs in inverse search graph G^{IS} until reaching the source node S (indicating a simple path) or one node in the walk (indicating a cyclic walk).

Lemma 1. *Let P be the path from a node v to a node u in search graph G^S. Then $Lab(u) - Lab(v) \geq Len(P)$, where $Lab(u)$ labels the node u and $Len(P)$ denotes the length of a path P.*

Proof. The proof is completed by mathematical induction on m (the number of arcs of path P). For $m = 1$, $Lab(u) - Lab(v) = Len((v, u))$ after node u is scanned, and $Lab(u) - Lab(v) > Len((v, u))$ after node v is re-scanned. For $m = k - 1$, suppose that Lemma 1 holds. For $m = k$, let node w be the parent of node u, and let P_1 be the path from node v to node w in search graph G^S. Then, $Lab(w) - Lab(v) \geq Len(P_1)$ and $Lab(u) - Lab(w) \geq Len((w, u))$ imply $Lab(u) - Lab(v) \geq Len(P)$, as claimed. □

Lemma 2. *If cycle C forms in search graph G^S, $Len(C) < 0$.*

Proof. Suppose that cycle C forms when node v is re-scanned. It implies that $Lab(v) < Lab^{old}(v)$, where $Lab^{old}(v)$ and $Lab(v)$ label node v before and after it is re-scanned, respectively. By Lemma 1, $Len(C) \leq Lab(v) - Lab^{old}(v) < 0$, as claimed. □

A negative cycle cannot be detected by a shortest path algorithm that collects local information only, like the parent and the label of a node, until it is traversed completely. It is assumed that the shortest path algorithm collects local information only through this paper. Consequently, a negative cycle containing source node S is encountered whenever source node S is re-scanned and vice versa. It implies that a negative cycle containing source node S can be easily located as soon as source node S is re-scanned. For conveniences' sake, it is also assumed, hereafter, that a network G has not a negative cycle containing source node S.

Theorem 1. *Network G has a negative cycle if and only if a negative cycle has ever formed in search graph G^S.*

Proof. Let the negative cycle $C : (c_0, c_1, \cdots, c_{n-1})$ has ever formed in search graph G_S. Clearly, the cycle $(c_0, c_1, \cdots, c_{n-1})$ is in a network G; it is a negative cycle. Conversely, let network $G(N, A)$ have a negative cycle. Consequently, the shortest path algorithm is not terminating. Without loss of generality, there are three assumptions:

A1 An arc in A weighs an integer. And, $w_{min}(< 0)$ and $w_{max}(> 0)$ are the minimum and maximum weights of arcs in A, respectively.

A2 $Lab(i)$ is initially set to $(|N| - 1) \times w_{max}$ for all nodes i except that $Lab(1)$, the label of source node S, is initially set to 0.

A3 $\Psi = \{(x_1, \cdots, x_{|N|}) | \forall i, (|N| - 1) \times w_{min} \leq x_i \leq (|N| - 1) \times w_{max}\}$.

Suppose that a negative cycle never forms in search graph G^S. Let P_i be the path from source node S to node i. Clearly, $(|N| - 1) \times w_{min} \leq Len(P_i)$. By Lemma 1, $Len(P_i) \leq Lab(i) - Lab(1)$. Thus $(|N| - 1) \times w_{min} \leq Lab(i)$. Besides, $Lab(i) \leq (|N| - 1) \times w_{max}$ since $Lab(i)$ is non-increasing during the execution of the shortest path algorithm. Thus $(|N| - 1) \times w_{min} \leq Lab(i) \leq (|N| - 1) \times w_{max}$. Therefore, the shortest path algorithm can be considered as a rewrite system R over Ψ. When the node i is scanned, $((Lab(1), \cdots, Lab^{old}(i), \cdots, Lab(|N|)) \Rightarrow_R (Lab(1), \cdots, Lab(i), \cdots, Lab(|N|)))$ implies $((Lab(1), \cdots,$

$Lab^{old}(i), \cdots, Lab(|N|)) >_{lex} (Lab(1), \cdots, Lab(i), \cdots, Lab(|N|)))$. It is clear that the lexicographic ordering ($\Psi >_{lex}$) is well-founded, and the shortest path algorithm is terminating by Theorem 5 of Dershowitz [3]. It is a contradiction. □

Lemma 3. *The search graph G^S is either a tree or a disconnected graph that contains at least one negative cycle.*

Proof. It is first proven that $|N^S| = |A^S| + 1$ by mathematical induction on m (the total times of nodes being scanned). For $m = 1$, source node S is the only node in search graph G^S. It implies $|N^S| = 1$ and $|A^S| = 0$. For $m = k - 1$, suppose that $|N^S| = |A^S| + 1$ holds. For $m = k$, two cases merit discussion when the node v is scanned:

1. The node v has never been scanned.
2. The node v has ever been scanned.

In case 1, both $|N^S|$ and $|A^S|$ increase by 1. In case 2, both $|N^S|$ and $|A^S|$ remain unchanged. And, $|N^S| = |A^S| + 1$ in both cases. Next, the search graph G^S is a tree, or a disconnected graph containing at least one cycle. This is because the in-degree of each node in search graph G^S is at most 1. If the search graph G^S is a disconnected graph, it contains at least one negative cycle by Lemma 2. □

Theorem 2. *A network G has a negative cycle if and only if the search graph G^S has ever disconnected.*

Proof. Let the search graph G^S has ever disconnected. Consequently, it has ever contained at least one negative cycle, say C, by Lemma 3. It implies that network G has a negative cycle C. Conversely, let network G have a negative cycle. Consequently, a negative cycle has ever formed in search graph G^S by Theorem 1. It implies that search graph G^S has ever disconnected by Lemma 3. □

Definition 4. The search graph G^S is a disruptive search graph G^{DS} if and only if it remains disconnected once it disconnects.

The disruptive search graph G^{DS} remains disconnected after a negative cycle is encountered. Thus the negative-cycle detection can periodically check whether disruptive search graph G^{DS} disconnects.

3. A Class of Shortest Path Algorithms with Disruptive Search Graphs

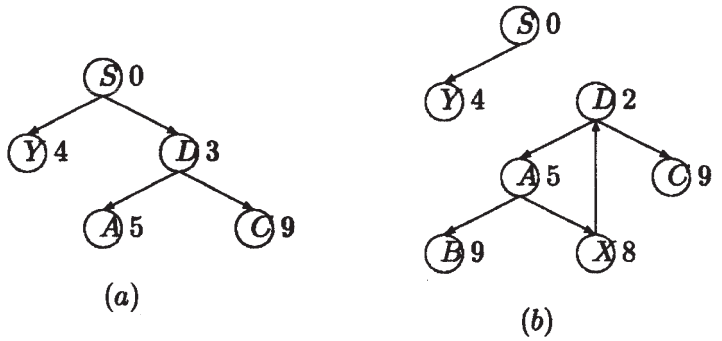

Figure 2: The search graph constructed when algorithm MD searches the network in Figure 1. (a) and (b) show the search graphs when node D is examined and when node D is first scanned by node X, respectively. The number labels the corresponding node.

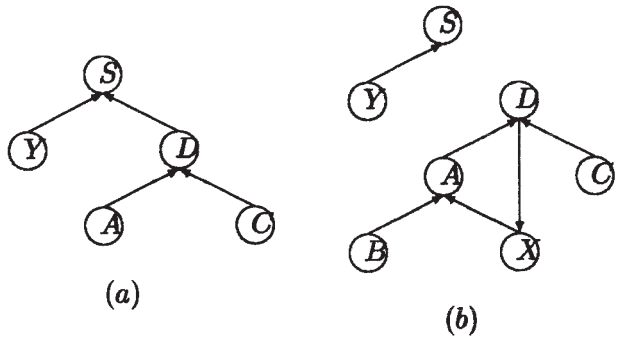

Figure 3: The inverse search graph constructed when algorithm MD searches the network in Figure 1. (a) and (b) show the inverse search graph when node D is examined and when node D is first scanned by node X, respectively.

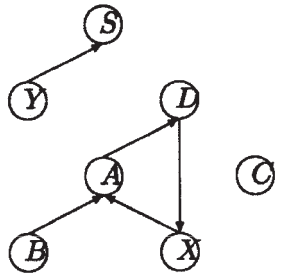

Figure 4: The parent walks of nodes Y and B in the inverse search graph in Figure 3 (b). $PW(Y) : Y \to S$ and $PW(B) : B \to A \to D \to X \to A$.

In this section, Dijkstra's algorithm and Bertsekas's SLF algorithm are slightly modified to construct disruptive search graphs. Dijkstra's algorithm finds shortest paths from source node S to all other nodes in a non-negative-weighted network by examining the closest non-examined node from source node S. In contrast with Dijkstra's label-setting method permanently labeling the node closest to source node S, the label-correcting method [7] however labels one node several times, and is able to find shortest paths in a non-negative-cycled network. Clearly, Dijkstra's algorithm can be modified (algorithm MD) to use the label-correcting method instead by ordering the scanned nodes for examination, and after, it is able to search a negative-weighted network and has disruptive search graph G^{DS} as shown in Theorem 3. Figures 2 and 3 show the search graph and the inverse search graph when algorithm MD searches the network in Figure 1, respectively. Figure 4 shows the parent walks of nodes Y and B. $PW(Y)$ is a simple path, and $PW(B)$ is a cyclic walk.

Theorem 3. *Algorithm MD has disruptive a search graph G^{DS}.*

Proof. There are four assumptions:

A1 The network G has a negative cycle. (Otherwise, search graph G^S never disconnects.)

A2 The search graph G^S is connected until node u scans node v. A negative cycle $C : (v, \cdots, w, u, v)$ forms when node u scans node v.

A3 The node y has smallest label among all nodes z with out-degree larger than 0 and connected with source node S.

A4 The node y waits for examination in queue Q when node u scans node v.

If node y does not existing, all nodes with out-degree larger than 0 and waiting for examination in queue Q are disconnected with source node S. It implies that search graph G^S will not be connected after. If node y is existing, there are two claims:

1. For all nodes p in cycle C, $Lab(p) \leq Lab(y)$ before node u scans node v.
2. The node z will not be examined when it is connected with source node S.

As for Claim 1, it is first proved that $Lab(u) \leq Lab(y)$. Suppose that $Lab(u) > Lab(y)$. Node y will be examined before node u. It implies that the node y does not wait for examination in queue Q. It contradicts A4. Next, it is proved that $Lab(w) \leq Lab(y)$ for node w, the parent of node u in the search graph G^S. Suppose that $Lab(w) > Lab(y)$. Let node x be the parent of node y in search graph G^S. If $Lab(w) > Lab(y)$ and $Lab(w) > Lab(x)$, both nodes x and y are examined before node w. It contradicts A4. If $Lab(w) > Lab(y)$ and $Lab(w) \leq Lab(x)$, $Lab(u) \leq Lab(y) < Lab(w) \leq Lab(x)$. It implies that node u is examined before node x. It contradicts A3. Thus $Lab(w) \leq Lab(y)$. The proof of Claim 1 is completed by repeating the same argument for the other nodes along $PW(w)$ one by one till node v is reached. As for Claim 2, it is trivial that $Lab(p) \leq Lab(z)$ for all nodes p in cycle C. Since node v in cycle C, $Lab(v) < Lab^{old}(v) \leq Lab(z)$ when node v is scanned. It implies that node v will be injected into queue Q before node z. And, the node with out-degree > 0 and selected for examination is always disconnected with source node S by repeating the same argument for any other node along cycle C. Therefore, search graph G^S remains disconnected once it disconnects, as desired. □

Moreover, Bertsekas's SLF algorithm is a low-overhead approximation of Dijkstra's algorithm. It places the newly scanned node v at the end of queue Q, if the label of node v is larger than that of the head node of queue Q; node v is inserted into the head of queue Q, otherwise. Besides, algorithm SLF is combined with threshold algorithm (STH) [9] that is faster than almost every known shortest path algorithm. Algorithm STH is slightly modified (algorithm $MSTH$) to have disruptive search graph G^{DS}. Algorithm $MSTH$ finds shortest paths as fast as algorithm STH in our empirical studies.

Algorithm $MSTH$

1. Insert the source node S into the queue Q_1.

2. Remove node u from the head of queue Q_1, and examine node u.
3. For each node v scanned by node u, repeat Steps (a) – (c).
   (a) If node v is in queue Q_1 (or Q_2), it has ever been examined, and $Lab(v) \leq threshold$, then remove the node v from queue Q_1 (or Q_2) and go to Step (c).
   (b) If node v is in queue Q_1 or Q_2, go to Step 3.
   (c) Insert the node v into
      i. the head of queue Q_1 if $Lab(v) \leq Lab(x)$ or ($Lab(x) < Lab(v) \leq threshold$ and node v has ever been examined),
      ii. the tail of queue Q_1 if $Lab(x) < Lab(v) \leq threshold$,
      iii. the head of queue Q_2 if $threshold < Lab(v) \leq Lab(y)$, and
      iv. the tail of queue Q_2 if $Lab(y) < Lab(v)$,

   where node x is at the head of queue Q_1, the node y is at the head of queue Q_2, and $Lab(x)$ and $Lab(y)$ are set to $threshold$ if queues Q_1 and Q_2 are empty, respectively.
4. If both queues Q_1 and Q_2 are empty, stop.
5. If queue Q_1 is empty, then adjust $threshold$, remove every node v satisfying $Lab(v) \leq threshold$ from queue Q_2, and insert node v into
   (a) the head of queue Q_1 if $Lab(v) \leq Lab(x)$, and
   (b) the tail of queue Q_1 otherwise,

   where node x is at the head of queue Q_1.
6. Goto Step 2.

Instead of placing node v at the tail of queue Q_1 when $Lab(x) < Lab(v) \leq threshold$ in algorithm STH, algorithm $MSTH$ places node v at the head of queue Q_1 if the node v has ever been examined before. It is also noted that $threshold$ is non-decreasing during the execution of algorithm MSTH [6].

Theorem 4. *Algorithm $MSTH$ has disruptive search graph G^{DS}.*

Proof. Suppose that the search graph G^S becomes disconnected and node z is connected with source node S when node u scans node v. It suffices to prove that node z will not be examined when it is connected with source node S. There are two claims as follows:

1. The node v will not be inserted into the tail of queue Q_1.

2. The node v will not be placed in queue Q_2.

Claim 1 holds because the node v has ever been examined at least once. Claim 2 holds because $Lab(v) < Lab^{old}(v) \leq threshold^{old} \leq threshold$, where $threshold^{old}$ is the threshold value when the node v was examined last time. And, the proof is completed by repeating the same argument for the other nodes along cycle C. □

4. Concluding Remarks

In this paper, a class of shortest path algorithms whose search graphs remain disconnected after negative cycles are encountered is introduced. A simple approach to detect negative cycles in a given network is to periodically check whether the search graph disconnects along with the execution of these algorithms. Thus, the time delay and the checking frequency can reach an arbitrarily compromise. In addition, Dijkstra's and Bertsekas's shortest path algorithms are slightly modified without lowering their corresponding efficiency to belong to this class.

References

[1] R.E. Bellman, On a routing problem, *Quart. Appl. Math.*, 16 (1958), 87-90.

[2] D.P. Bertsekas, A simple and fast label correcting algorithm for shortest paths, *Networks*, 23 (1993), 703-709.

[3] N. Dershowitz, Termination of rewriting, *J. Symbolic Computation*, 3 (1987), 69-116.

[4] R. Dial, F. Glover, D. Karney, D. Klingman, A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees, *Networks*, 9 (1979), 215-248.

[5] E.W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.*, 1 (1959), 269-271.

[6] W. Domschke, Two algorithms to detect negative cycles in a valued graph, *Computing*, 11 (1973), 124-136.

[7] S.E. Dreyfus, An appraisal of some shortest-path algorithms, *Operations Res.*, 17 (1969), 395-411.

[8] M. Florian, P. Roberts, On the efficiency of a direct search method to locate negative cycles in a network, *Management Sc.*, 19 (1972), 333-335.

[9] F. Glover, D. Klingman, N. Phillips, A new polynomially bounded shortest path algorithms, *Operations Res.*, 33 (1985),65-73.

[10] D. Goldfarb, J. Hao, S.R. Kai, Shortest path algorithm using dynamic breadth-first search, *Networks*, 21 (1987), 29-50.

[11] D.B. Johnson, Efficient algorithms for shortest paths in a network, *J. Assoc, Comput. Mach.*, 24 (1977), 1-13.

[12] R.M. Karp, Note: a characterization of the minimum cycle mean in a digraph, *Discrete Math.*, 23 (1978), 309-311.

[13] M. Klein, R.K. Tibrewala, Finding negative cycles, *INFOR-Canada J. Operational Res. and Information Processing*, 11 (1973), 59-65.

[14] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, New York: Holt, Rinehart and Winston (1976).

[15] T. Lengauer, Efficient algorithms for the constraint generation for integrated circuit layout compaction, In: *Proc. of 9-th Workshop on Graph-theoretic Concepts in Computer Science* (1983), 219-230.

[16] D. Maier, An efficient method for storing ancestor information in trees, *SIAM Journal of Computing*, 8 (1979), 599-618.

[17] C. Papadimitriou, Steiglitzk, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall (1982).

[18] P. Spirakis, A. Tsakalidis, A very fast, practical algorithm for finding a negative cycle in a digraph, *Automata, Languages and Programming: 13-th international colloquium* (1986), 397-406.