International Journal of Management Science
Vol 13, No 1, May 2007

Essential Arcs of a Directed Acyclic Graph

Eesuk Chung*

Dept. of Industrial Engineering, Korea Advanced Institute of Science and Technology,
373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, Korea

(Received Apr. 2007; Revised May 2007; Accepted May 2007)

ABSTRACT

Among many graphs, directed acyclic graph (DAG) attracts many researchers due to its nice property of existence of topological sort. In some classic graph problems, it is known that, if we use the aforementioned property, then much efficient algorithms can be generated. So, in this paper, new algorithm for the all-pairs shortest path problem in a DAG is proposed. While the algorithm performing the iteration, it identifies the set of essential arcs which requires in a shortest path. So, the proposed algorithm has a running time of $O(m^*n+m)$, where m, n and m^* denote the number of arcs, number of node, and the number of essential arcs in a DAG, respectively.

Keywords: Directed Acyclic Graph, All-pairs Shortest Paths, Essential Arcs

1. Introduction

Let G be a directed graph having n nodes and m arcs with arbitrary arc costs. The cost of an arc from node u to node v is c_{uv}, and the distance of the shortest path from u to v in G is denoted d_{uv}. An arc (u, v) is *essential* if $d_{uv} = c_{uv}$. Let m^* denote the number of essential arcs in G. It is sufficient to find the shortest-paths in the subgraph G^* consisting of the set of essential arcs instead of solving the shortest-paths problem in G.

Karger *et al.* [4] and McGeoch [7], independently, presented algorithms that solve the all-pairs shortest paths problem for general graphs and find the set of essential arcs at once. Their algorithms run in a running time of $O(m^*n+n^2\log n)$ if the Fibo-

* Email: eesuk@kaist.ac.kr

nacci heap is used as a priority queue in the implementation of the Dijkstra's algorithm.

This paper presents an algorithm for identifying the essential arcs of a directed acyclic graph (DAG) in $O(m^*n+m)$ by exploiting the structural properties unique to a DAG. Some graph problems are easier to solve if the underlying graph is a DAG. Consider the single-destination shortest paths problem. It allows an $O(mn)$ algorithm for arbitrary arc costs [1, 6] and $O(m+n\log n)$ one for nonnegative arc costs [2]. In contrast, it can be solved in $O(m)$ for a DAG with arbitrary arc costs [5]. A DAG has a topological sort (of its nodes), an ordering of the nodes such that, if there is path from node u to node v, then v appears after u in the ordering. The single-destination shortest paths problem in a DAG is simpler than in a general graph because of the existence of the topological sort. In this paper, we investigate the benefits of the property in identifying the essential arcs in the all-pairs shortest paths problem.

2. Structural Properties of DAGs

Let $G=(V,E)$ be a topologically sorted DAG: that is, if $(i,j)\in E$ then $i<j$. If there exists a path between any two nodes i and $j(i<j)$ in G, say, $i\rightarrow k_1\rightarrow k_2\rightarrow\cdots\rightarrow k_l\rightarrow j$, then the nodes on the path also have a relationship $i<k_1<k_2<\cdots<k_l<j$. From this property, we get the following result.

Lemma 1. *Define $G[i..j]$ for two nodes i and $j(i<j)$ by a subgraph induced from a node set $\{i,i+1,\cdots,j-1,j\}$ in G. Consider any two nodes p and $q(p<q)$ in the subgraph $G[i..j]$ of a topologically sorted DAG G. The shortest path from the node p to the node q in G is included in $G[i..j]$.*

Proof. Let us assume that the shortest path P from p to q contains some nodes not belonging to $G[i..j]$. Among such nodes, denote the last node on the shortest path P by r. Hence we have $j<r$. Since p reaches r, $p<r$ holds. Denote the node next to r in the shortest path P by k. Since $k\in G[i..j]$, $r<k<j$. Combining these inequalities gives $j<r<k<j$, which is a contradiction. □

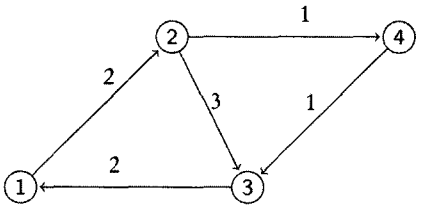
Define $G_k = G[1..k]$ and observe that $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_{n-1} \subseteq G_n = G$. We also define $E_k, E_k^*, \delta_k = \{(j,k) : (j,k) \in E\}$ as the arc set of G_k, the essential arc set of G_k and the incoming arcs to the node k of G, respectively. Then we have the following incremental relationship (1) and (2).

$$E_k = E_{k-1} \cup \delta_k, \quad E_{k-1} \cap \delta_k = \phi \tag{1}$$

Theorem 1. *If G is a topologically sorted DAG then*

$$E_k^* = E^* \cap E_k$$

Proof. The incremental relationship (1) and Lemma 1 implies that the shortest path in G_k is preserved even in the extended graph G_{k+1}, which supports that the essential arcs in G_k are the essential arcs of G which also belongs to G_k. □



$$E_3^* = \{(1,2),(2,3),(3,1)\}$$
$$E^* = E_4^* = \{(1,2),(2,4),(4,3),(3,1)\}$$
$$E_3^* \neq E^* \cap E_3$$

Figure 1. Theorem 1 is not true in a general graph which is not a DAG

That is, in a DAG, a path being shortest in a subgraph G_k is the shortest one in the original graph G, which is not true in general (see Figure 1). This property plays a key role in the identification of the essential arcs by an incremental way.

Theorem 2. *Let G be a topologically sorted DAG then*

$$E_k^* = E_{k-1}^* \cup \delta_k^*, \quad E_{k-1}^* \cap \delta_k^* = \phi \tag{2}$$

where $\delta_k^* = E^* \cap \delta_k = \{(i,k) \in \delta_k : d_{ik} = c_{ik}\}$ and d_{ik} is defined by the shortest distance from the node i to the node k.

Proof. $E_k^* = E^* \cap E_k = E^* \cap (E_{k-1} \cup \delta_k) = (E^* \cap E_{k-1}) \cup (E^* \cap \delta_k) = E_{k-1}^* \cup \delta_k^*$ □

3. Algorithm

We can solve all-pairs shortest paths problem by running single-source shortest paths algorithm once from each node. Johnson [3] approaches the problem in this way for the general graph. We apply the same idea for the topologically sorted DAGs. The single-destination shortest paths from each node to k-th node can be solved by considering only the arcs in G_k (instead of all arcs in G) by Lemma 1.

algorithm ASP-DAG(G)
1 $d_{kk} \leftarrow 0$ **for** all k
2 **for** $k \leftarrow 2$ **to** n
3 **for** $i \leftarrow k-1$ **downto** 1
4 $d_{ik} \leftarrow \min\{c_{ij} + d_{jk} : (i,j) \in E_k\}$

The lines 3-4 compute the single-destination shortest paths to node k in G_k and takes $O(|E_k|) = O(m)$ time. Hence the running time of ASP-DAG is $\sum_{k=2}^{n} O(|E_k|) = O(mn)$.

From Theorem 1 we know that a nonessential arc of G_k is not necessary to compute G_v^*, $v \geq k$. By removing nonessential arcs the computational effort is saved. By Theorem 2 we can do such computation in an incremental way. To identify E_k^* based on E_{k-1}^*, consider the graph $\tilde{G}_k$ induced by the essential arcs G_{k-1}^* and the incoming arc set δ_k. Since $\tilde{E}_k^* \cap E_{k-1}^* = E_{k-1}^*$ and $\setminus \tilde{E}_k^* \setminus E_{k-1}^* \subseteq \delta_k$, we have to find only the additional essential arcs $\delta_k^* = \{(i,k) \in \delta_k : d_{ik} = c_{ik}\}$.

By the incremental relationship (2), ASP-DAG is reformulated as follows.

algorithm ASP-DAG-ESSENTIAL(G)
1 $d_{kk} \leftarrow 0$, $\delta_k^* \leftarrow \phi$ **for** all k
2 $E_1^* = \phi$
3 **for** $k \leftarrow 2$ **to** n
4 **for** $i \leftarrow k-1$ **downto** 1
5 $d_{ik} \leftarrow \min\{c_{ij} + d_{jk} : (i,j) \in E_{k-1}^* \cup \delta_k\}$
6 **if** $c_{ik} \in \delta_k$ **and** $d_{ik} = c_{ik}$ **do** $\delta_k^* \leftarrow \delta_k^* \cup \{(i,k)\}$
7 $E_k^* \leftarrow E_{k-1}^* \cup \delta_k^*$

The lines 4-6 compute the single-destination shortest paths to node k in $\tilde{G}_k$ and identify the additional essential arcs δ_k^*.

Theorem 2. *The running time of ASP-DAG-ESSENTIAL is* $O(m^* n + m)$.

Proof. The line 5 takes $O(|E_{k-1}^*| + |\delta_k|)$ time for each k. The lines 6 takes $O(|\delta_k|)$ time for each k. Hence the total complexity is

$$\sum_k O(|E_{k-1}^*| + 2|\delta_k|) \le \sum_k O(|E^*| + 2|\delta_k|) = O(m^* n + m)$$ □

4. An Example

Consider a DAG having 6 nodes and arc costs given by the Figure 2. The arc costs associated with non-essential arcs are asterisked. We identify G_5^* using G_4^*. If we compute the shortest path from all other nodes (less than 5) to the node 5 on the subgraph $\tilde{G}_5 = G_4^* \cap \delta_5$, we get a shortest path tree. See Figure 2.

	1	2	3	4	5	6
1	·	2	6	6*	9*	9
2	·	·	7	1	6*	7
3	·	·	·	2	6*	5
4	·	·	·	·	1	9*
5	·	·	·	·	·	7
6	·	·	·	·	·	·

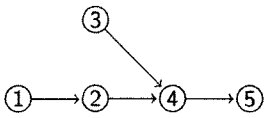


Figure 2. Arc cost matrix and shortest path tree of $\tilde{G}_5$

Arcs (1, 5), (2, 5), and (3, 5) are non-essential arcs because they are not on the shortest path. That is, these three arcs are not considered in the next step identifying G_6^*. By excluding the non-essential arcs in the preceding steps, we save the amount of computations. In the considered example, 10 arcs are essential among 15 original arcs. The proposed algorithm is likely to be fast in practice even though $m^* = O(m)$, because several probabilistic analyses (refer to [7] and references therein) show that $m^* = O(n \log n)$ under some distributional assumptions on the arc costs.

References

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

[2] Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik* 1 (1959), 269-271.

[3] Johnson, D. B., "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM* 24, 1 (1977), 1-13.

[4] Karger, D. R., D. Koller, and S. J. Phillips, "Finding the hidden path: Time bounds for all-pairs shortest paths," *SIAM Journal on Computing* 22, 6(1993), 1199-1217.

[5] Lawler, E. L., *Combinatorial Optimization: Networks and Matrioids*, Holt, Rinehart, and Winston, 1976.

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

[7] McGeoch, C. C., "All-pairs shortest paths and the essential subgraph," *Algorithmica* 13 (1995), 426-441.