A special case of the dynamization problem for least cost paths

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Theoretische Informatik

Davide Crippa

A Special Case of the Dynamization Problem for Least Cost Paths

March 1991

155

Authors´addresses:

Institut für Theoretische Informatik
ETH-Zentrum
CH-8092 Zürich
e-mail: crippa@inf.ethz.ch

Abstract

Given a digraph and a cost function on its edges, we want to develop a structure supporting questions of the type: what is the cost of the least cost path from a given origin r to each node v if the cost of each edge is increased by the same positive constant δ. We will see that an efficient structure can be generated as by-product of the Bellmann-Ford algorithm for least cost paths; the main feature of this structure is that for any choice of δ such questions can be answered in time $O(\log \lambda)$, where λ is the length of the longest least cost path in the digraph.

Contents

1 Introduction

In working with large networks it is sometimes necessary to modify some of their characteristics; then, in order to reduce the time needed for finding the solution of a given problem, it is advisable to update the previously found solutions instead of computing everything from scratch again. This is what we tried to do for our problem, which could be seen as a special case of the dynamization problem, solved by Even/Shiloach [2] and Ibaraki/Katoh [5] for transitive closures, by La Poutré/van Leeuwen [7] for transitive closures and reductions, by Frederickson [4] for minimum spanning trees and by Rohnert [10] for all-pairs least cost paths.

The problem can be formalized as follows: given a digraph $G = (V, E)$, $|V| = n, |E| = m$, and a cost function $C : E \rightarrow I\!R$, which does not imply negative cost cycles, we want to compute

$$\textit{LeastCost}(r, v, \delta) = \text{cost of the least cost path from a given origin } r \text{ to the node } v \text{ when the cost of each edge in the graph is increased by the positive constant } \delta$$

for different choices of δ without having to run a least cost path algorithm everytime with a new cost function. This will be obtained by generating in time $O(m\,\lambda)$ a structure that will allow us to answer every $\textit{LeastCost}(r, v, \delta)$ in time $O(\log \lambda)$, where λ is the length of the longest least cost path in G.

2 The Main Idea

Throughout this paper we suppose that a given origin r is specified, from which all the least cost paths are computed; therefore, whenever we speak about a path to v we mean a path starting at r, and least cost path trees are also meant to be rooted at this node.

As usual we denote by $adj^+(v)$ (resp. $adj^-(v)$) the set of successors (resp. predecessors) of the node v in G. Further we will write $P[e,c]$ for a path with e edges whose total cost is c , $LCP(v)$ for the least cost path to the node v and $LCP_k(v)$ for the least cost path to the node v of length not exceeding k ; we define $G(\delta)$ as the graph obtained from G by adding the positive constant δ to the cost of each edge, and we call δ the *offset* of the new graph with respect to the original one. Finally, by *updating an offset* δ we mean computing the least cost path tree of the graph $G(\delta)$.

Property 1 *The number of edges in the $LCP(v)$ is a non increasing function with respect to the offset, i.e. when updating an offset the new $LCP(v)$ will have at most as many edges as the old one.*

Proof. Let $P[e_1,c_1]$ be the $LCP(v)$; for any other path $P[e,c]$ to v we must have $c \geq c_1$. When we update an offset δ these paths become respectively $P[e_1, c_1+\delta e_1]$ and $P[e, c+\delta e]$; if now another path becomes the $LCP(v)$ we must have $c+\delta e < c_1+\delta e_1$, which is possible only if $e < e_1$. □

Property 2 *Given two paths $P[e_1,c_1]$ and $P[e_2,c_2]$ to v with $e_1 > e_2$ and $c_1 < c_2$, the latter will become less expensive than the former at the offset $\delta = (c_2-c_1)/(e_1-e_2)$.*

Proof. Immediate, as the costs of the two paths at an offset δ are respectively $c_1+\delta e_1$ and $c_2+\delta e_2$. □

Property 3 *There exists an offset Δ such that in $G(\Delta)$ the length of the $LCP(v)$ is equal to the distance between v and r. Further the least cost path tree of $G(\delta)$ will be the same as the one of $G(\Delta)$ for all $\delta > \Delta$.*

Proof. It follows immediately from Properties 1 and 2. □

Theorem 1 *For every $\delta \geq 0$ and $v \epsilon V$ there exists a $k \epsilon I\!N$ such that*

$$LCP(v) \text{ in } G(\delta) = LCP_k(v) \text{ in } G$$

Proof. Moreover, if the $LCP(v)$ in $G(\delta)$ has length k, we can show that this path is the same as the $LCP_k(v)$ in G. Let $P[k, c]$ be the path in G that will be the $LCP(v)$ in $G(\delta)$, and let another path $P[e', c']$ be the $LCP_k(v)$ in G. Then $e' \leq k$ and $c' < c$, so that in $G(\delta)$ we will obtain $c + k\delta > c' + e'\delta$ for the costs of the two paths, in contradiction with our assumptions. □

Theorem 2 *Let $P_i[e_i, c_i]$, $i = 1, 2, 3$, be three paths to a node v in G with $e_1 > e_2 > e_3$ and $c_1 < c_2 < c_3$; further let $\delta_{1,2}$, $\delta_{1,3}$ and $\delta_{2,3}$ be the offsets at which P_2 becomes less expensive than P_1, respectively P_3 less expensive than P_1 and P_3 less expensive than P_2.*
Then, if $\delta_{1,2} < \delta_{1,3}$, it must also be $\delta_{1,2} < \delta_{2,3}$

Proof. The following inequalities are all equivalent:

$$\begin{aligned}
\delta_{1,2} &< \delta_{1,3} \\
(c_2 - c_1)(e_1 - e_3) &< (c_3 - c_1)(e_1 - e_3) \\
c_2e_1 - c_2e_3 + c_1e_3 &< c_3e_1 - c_3e_2 + c_1e_2 \\
-c_1e_2 - c_2e_3 + c_1e_3 &< c_3e_1 - c_3e_2 - c_2e_1 \\
(c_2 - c_1)(e_2 - e_3) &< (c_3 - c_2)(e_1 - e_2) \\
\delta_{1,2} &< \delta_{2,3}
\end{aligned}$$

□

We now have enough arguments to state the main idea of the algorithm, which consists in finding for each $v \in V$ all the offsets δ for which

$$LCP(v) \text{ in } G(\delta) \neq LCP(v) \text{ in } G(\delta'), \text{ for all } 0 \leq \delta' < \delta$$

i.e., all the offsets at which the structure of the least cost path to v changes, which are the relevant data to compute $LeastCost(r, v, \delta)$ for all $\delta > 0$. In the algorithm we will use a list $BLCP(v)$ representing the least cost paths of bounded length to v **in** G. The construction of this list will be described in the next section.

Main Algorithm

(1) **forall** $v \in V$ **do**
(2) compute the list $BLCP(v)$;
(3) find out which of the paths in $BLCP(v)$ will become $LCP(v)$ in a $G(\delta)$ and at which δ;
(4) generate the structure to answer $LeastCost(r, v, \delta)$ for any $\delta \geq 0$
(5) **od;**
(6) **while not** stop **do** answer $LeastCost(r, v, \delta)$ **od.**

Main Algorithm

3 Least Cost Paths of Bounded Length

We now present the procedure which computes for each node v in the digraph G all the least cost paths to v of bounded length (Step (2) of the Main Algorithm). Each path will be stored in the list $BLCP(v)$ as record of two variables: its *cost* and its *length* in G [1].

BLCP

```
(1)  PQ ← [ r | 0 ] ;
(2)  forall u ϵ V do BLCP(u) ← [ ∞ | ∞ ] od;
(3)  BLCP(r) ← [ 0 | 0 ] ;
(4)  repeat
(5)        [ u | k ] ← the first record in PQ;
(6)        u_path ← the first record in BLCP(u);
(7)        forall v ϵ adj+(u) do
(8)           v_path ← the first record in BLCP(v);
(9)           if u_path.cost + C(u,v) < v_path.cost
(10)          then
(11)             new_v_cost ← u_path.cost + C(u,v);
(12)             new_v_length ← u_path.length + 1;
(13)             if v_path.length = new_v_length
(14)             then v_path.cost ← new_v_cost
(15)             else
(16)                put [ v | new_v_length ]
                    at the end of PQ;
(17)                put [ new_v_cost | new_v_length ]
                    at the begin of BLCP(v)
(18)             fi
(19)          fi
(20)       od
(21) until PQ = ∅
```

BLCP

[1] In each record we could also store the predecessor of the node in that path, so that we would be able to reconstruct in the usual way the corresponding path; as this is not very relevant to our analysis we will omit it.

In particular the list is constructed in such a way that the records are *ordered by decreasing number of edges in the path*, which will be useful in the next step of the Main Algorithm. Notice that the list contains only the different least cost paths of bounded length: this means that if $P[e_1, c_1]$ is the immediate predecessor of $P[e_2, c_2]$ in the list, then the latter is the $LCP_k(v)$ for all $e_2 \leq k < e_1$.

Let us denote by $c_v^{(k)}$ the cost of $LCP_k(v)$ in G. The presented procedure is a by-product of the Ford's algorithm [3], which was based on the equations

$$\begin{aligned} c_r^{(0)} &= 0 \\ c_v^{(0)} &= \infty \ \forall\, v \neq r \\ c_v^{(k+1)} &= \min \left\{ c_v^{(k)} \,,\ \{ c_u^{(k)} + C(u,v) \mid u \in adj^-(v) \} \right\} \ \forall\, v \in V \end{aligned}$$

It consists in introducing a priority-queue PQ (as developed by Ford [3] and Moore [9]) where we store each *node* for which a new, longer LCP of bounded length has been found together with the *length* of this path:

$$\begin{aligned} PQ^{(0)} &= \{r\} \\ PQ^{(k)} &= \{ v \in V \mid c_v^{(k)} < c_v^{(k-1)} \} \ \forall\, k = 1, .., n \end{aligned}$$

If now we order the Ford's equations differently we get the following procedure:

$$\begin{aligned} c_v^{(k+1)} &= c_v^{(k)} \ \forall\, v \in V \\ c_v^{(k+1)} &= \min \left\{ c_v^{(k+1)},\ c_u^{(k)} + C(u,v) \right\} \ \forall\, v \in adj^+(u),\ u \in PQ^{(k)} \end{aligned}$$

If finally we choose for PQ a first-in-first-out strategy, we automatically obtain the list $BLCP(v)$ ordered with respect to the *length*.

4 The Structure LeastCostTree

We now come to the description of the other steps of the Main Algorithm, which are developed in a single procedure.

As mentioned before, the list $BLCP(v)$ contains all the possible candidates for $LCP(v)$ in a $G(\delta)$ and by construction they are ordered by decreasing *length*; in particular the first path, say $P_0[e,c]$, is the $LCP(v)$ in G. Now we compare it with all the other candidates and, according to Property 2, we determine at which offset each of these would become less expensive than P_0; the minimum δ' over these values, corresponding say to the path $P'[e',c']$, is the smallest offset at which the structure of $LCP(v)$ changes. This means that for $0 \leq \delta < \delta'$ the least cost path to v in $G(\delta)$ is given by P_0, and therefore it follows immediately that $LeastCost(r,v,\delta) = c + e\delta$ for all $0 \leq \delta < \delta'$. Continuing, we compare now P' with all the candidates which are shorter (because of Proposition 1), i.e. with all its successors in $BLCP(v)$, and again we compute the minimum δ'' over the possible offsets; at this point Theorem 2 makes sure that $\delta'' > \delta'$, and therefore for $\delta' \leq \delta < \delta''$ the $LCP(v)$ in $G(\delta)$ will be P'. And so on.

LeastCostTree

(1) **forall** $v \in V$ **do**
(2) $\quad LeastCostTree(v) \leftarrow$ empty tree;
(3) $\quad P_0 \leftarrow$ the first record in $BLCP(v)$;
(4) $\quad$ insert $\boxed{P_0.cost \mid P_0.length \mid 0}$ in $LeastCostTree(v)$;
(5) $\quad$ **while** P_0 not the last record in $BLCP(v)$ **do**
(6) $\quad\quad$ let P' be the record in $BLCP(v)$ such that

$$\delta_{P'} = \min \left\{ \frac{P.cost - P_0.cost}{P_0.length - P.length} \;\middle|\; P \text{ successor of } P_0 \right\}$$

(7) $\quad\quad$ insert $\boxed{P'.cost \mid P'.length \mid \delta_{P'}}$ in $LeastCostTree(v)$;
(8) $\quad\quad P_0 \leftarrow P'$;
(9) $\quad$ **od**;
(10) **od**

LeastCostTree

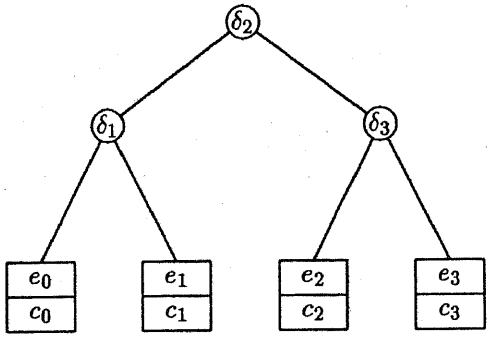
As the only relevant data needed to compute $LeastCost(r, v, \delta)$ are the offsets at which the structure of the $LCP(v)$ changes together with the length and the cost of these paths in G, we need a structure to store these values. For this purpose we assign to each node v a balanced search tree $LeastCostTree(v)$, which could be for instance a BB[α]-Tree (see Mehlhorn [8]). For this Data Structure the basic operations **Insert** and **Search** require time $O(\log N)$, where N is the number of data stored in the tree. Usually sets are stored in BB[α]-trees in node-oriented fashion; in our case, though, we will assign the offsets to the nodes and the other information regarding the paths to the leaves. If now we want to compute $LeastCost(r, v, \delta)$ we proceed in the following way:

LeastCost(r, v, δ)

```
(1)  node ← root of LeastCostTree(v);
(2)  while node ≠ leaf do
(3)     if δ < node.key
(4)     then go to the left son of node
(5)     else go to the right son of node
(6)     fi;
(7)  od;
(8)  LeastCost(r, v, δ) ← leaf.cost + leaf.length · δ
```

LeastCost(r, v, δ)

To explain better the last steps let us make an example. Suppose that the structure of $LCP(v)$ in $G(\delta)$ changes at the three different offsets $\delta_1 < \delta_2 < \delta_3$ and let $P_i[e_i, c_i]$, $i = 1, 2, 3$, be the corresponding paths to v; further let $P_0[e_0, c_0]$ be the $LCP(v)$ in the originary $G = G(0)$. Then we would store these values in the following tree:



To compute $LeastCost(r, v, \delta)$, say for $\delta_1 \leq \delta < \delta_2$, we would start at the root δ_2 of the tree, go to its left son δ_1 and reach its right leaf. This leaf contains, as explained in a previous discussion, the characteristics e_1 and c_1 of the path in G that will become $LCP(v)$ in $G(\delta)$, and therefore

$$LeastCost(r, v, \delta) = c_1 + e_1 \cdot \delta$$

5 Concluding Remarks and Open Questions

The complexity of the whole algorithm can be set together by the following:

- to compute *BLCP* we need time $O(m\ \lambda)$, where λ is the length of the longest least cost path in G, because each node will be removed from PQ at most λ times, as shown in [9];
- further to compute *LeastCostTree* we have a complexity of $O(n\ \lambda\ \log\lambda)$: for each node v the steps (6)-(10) cost at most successively $((\lambda - i)\ \log i)$, $i = 1, .., \lambda - 1$;
- finally to answer *LeastCost*(r, v, δ) we need time $O(\log\lambda)$, because, according to Property 1, λ is an upper bound for the number of offsets at which the structure of $LCP(v)$ changes.

Our empirical results, obtained by running the algorithm on a weighted random digraph of the class $D_{n,p,c}$ (as defined in [1]), indicate that λ is in the order of magnitude of $\log n$. Is there a mathematical proof for this ?

An upper bound for the total number of offsets *Noff* is $n\lambda$, and this would show, if our conjecture about λ is exact, that *Noff* does not grow linearly with the number of edges. This again agrees with our empirical results.

Let us specify the concepts of *internal* resp. *external path length of a tree*: these are defined to be the sum, taken over all interior nodes respectively leaves, of the lengths of the paths from these nodes to the root (see also [6]). Another upper bound for *Noff* is now given by the difference between the sum of internal and external path lengths in the LCP-Tree and in the BFS-Tree in G; in fact by Property 1 at every offset there is at least an *LCP* whose length will decrease by at least 1, and further by Property 3 the sum of internal and external path lengths in the LCP-Tree after the biggest offset will be the same as the one in the BFS-Tree. Using the results about the diameter of random graphs obtained by Spirakis [11] we could say that the sum for BFS-Tree should be in average $O(n\ \log n)$, but again we do not know about any result concerning the sum for LCP-Trees.

References

[1] D.ANGLUIN AND L.VALIANT. *Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings.* J. Computers and System Sciences 18 (1979) 155-193.

[2] S.EVEN AND Y.SHILOACH. *An On-line Edge-Deletion Problem.* JACM 28-1 (1981).

[3] L.J.FORD JR. *Network Flow Theory.* Rand Corporation Report P-293 (1956).

[4] G.N.FREDERICKSON. *Data Structures for On-line Updating of Minimum Spanning Trees.* CACM (1983).

[5] T.IBARAKI AND N.KATOH. *On-line Computation of Transitive Closures of Graphs.* Information Processing Letters 16 (1983) 95-97.

[6] R.KEMP. *Fundamentals of the Average Case Analysis of Particular Algorithms.* Wiley-Teubner (1984) 124.

[7] J.A. LA POUTRÉ AND J. VAN LEEUWEN. *Maintenance of Transitive Closures and Transitive Reductions of Graphs.* Lecture Notes in Computer Science 314 (1987) 106-120.

[8] K.MEHLHORN. *Data Structures and Algorithms 1/2.* EATCS, Springer (1984).

[9] E.F.MOORE *The Shortest Path through a Maze.* International Symposium on the Theory of Switching 1957, Harvard University Cambridge MA (1959).

[10] H.ROHNERT. *A Dynamization of the All Pairs Least Cost Path Problem.* Lecture Notes in Computer Science 182 (1985) 279-286.

[11] P.SPIRAKIS *The Diameter of Connected Components of Random Graphs.* Lecture Notes in Computer Science 246 (1986) 264-274.

Gelbe Berichte des Departements Informatik

135	Ch. Wieland	Two Explanation Facilities for the Deductive Data base Management System DeDEx
136	H.-J. Schek, M.H. Scholl, G. Weikum	From the KERNEL to the COSMOS: The Database Research Group at ETH Zurich
137	G. Weikum, Ch. Hasse A. Mönkeberg, P. Zabback	The COMFORT Project: A Comfortable Way to Better Performance (vergriffen)
138	J. Gutknecht	The Oberon Guide: System Release 1.2 (vergriffen)
139	P.E. Saylor, D.C. Smolarski	Implementation of an Adaptive Algorithm for Richardson 's Method (vergriffen)
140	N. Wirth	Die Programmiersprache Oberon (vergriffen)
141	M. Franz	The Implementation of MacOberon
142	M. Franz	MacOberon Reference Manual
143	N. Wirth	From Modula to Oberon (Revised Edition) (vergriffen)
144	J.L. Marais	The GADGETS User Interface Management System
145	J. Mössenböck	She: A Simple Hypertext Editor for Programs
146	H. E. Meier	Schriftgestaltung mit Hilfe des Computers Typographische Grundregeln
147	G. Weikum, P. Zabback, P. Scheuermann	Dynamic File Allocation in Disk Arrays
148	D. Degiorgi	A New Linear Algorithm to Detect a Line Graph and Output its Root Graph
149	A. Moenkeberg, G. Weikum	Conflict-Driven Load Control for the Avoidance of Data-Contention Thrashing
150	M.H. Scholl, Ch. Laasch M. Tresch	Updatable Views in Object-Oriented Databases
151	C. Szyperski	Write - An extensible Text Editor for the Oberon System
152	M. Bronstein	On Solutions of Linear Ordinary Difierential Equations in their Coefficient Field
153	G.H. Gonnet, D.W. Gruntz	Algebraic Manipulation: Systems
154	G.H. Gonnet, St.A. Benner	Computational Biochemistry Research at ETH