Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Discrete Mathematics 17 (2004) 81–86
Electronic Notes in DISCRETE MATHEMATICS
www.elsevier.com/locate/endm

Two-Levels-Greedy: a generalization of Dijkstra's shortest path algorithm

Domenico Cantone[1] Simone Faro[2]

Department of Mathematics and Computer Science, University of Catania
Viale A. Doria 6, 95125, Catania, Italy

The shortest path problem on weighted directed graphs is one of the basic network optimization problems. Its importance is mainly due to its applications in various areas, such as communication and transportation. Here we are interested in the single-source case. When the graph is not required to satisfy any particular restriction and negative weight edges can occur, the problem is solved by the Bellman-Ford-Moore algorithm [Bel58,For56,Moo59], whose complexity is $\mathcal{O}(|V||E|)$, with V and E denoting the sets of nodes and of edges, respectively. A more efficient solution due to Dijkstra [Dij59] is available when weights are restricted to non-negative values. Depending on the implementation used for maintaining a service priority queue, Dijkstra's algorithm has complexity $\mathcal{O}(|V|^2)$ (simple list), or $\mathcal{O}(|E|\log|V|)$ (standard binary heap), or $\mathcal{O}(|V|\log|V|+|E|)$ (Fibonacci heap [FT87]). Another case which can be solved very efficiently occurs when the underlying graph is acyclic. In such a case, by scanning the nodes in topological ordering, one can achieve a $\mathcal{O}(|V|+|E|)$ complexity.

In this note we present a natural generalization of Dijkstra's algorithm to the case in which negative weight edges are allowed, but only outside of any cycle. The resulting algorithm turns out to have the same asymptotic complexity of Dijkstra's algorithm and shows a linear behavior in the case of acyclic graphs. In fact, we will also see that our proposed algorithm compares

[1] Email: cantone@dmi.unict.it
[2] Email: faro@dmi.unict.it

1571-0653/$ – see front matter
doi:10.1016/j.endm.2004.03.019

very well in practice with the most efficient shortest path algorithms available, such as the ones due to Dial [Dia69], Pape [Pap74], Pallottino [Pal84], Glover *et. al.* [GGK84], and to Goldberg and Radzik [GR93].

1 Preliminaries

We begin by reviewing the relevant notations and terminology. A *directed graph* is represented as a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times V$ is a set of edges such that E does not contain any self-loop of the form (v, v). A *weight function* l on $G = (V, E)$ is any real function $l : E \to \mathbb{R}$. A *path* in $G = (V, E)$ is any finite sequence $(v_1, v_2, \ldots, v_k)$ of nodes such that (v_i, v_{i+1}) is an edge of G, for $i = 1, \ldots, k-1$. The weight function can be naturally extended over paths by putting $l(v_1, v_2, \ldots, v_k) = \sum_{i=1}^{k-1} l(v_i, v_{i+1})$. A *minimum weight path* (or shortest path) from u to v is a path in $G = (V, E)$ whose weight is minimum among all paths from u to v. Provided that v is reachable from u and that no path from u to v goes through a negative weight cycle, a minimum weight path from u to v exists; in such a case we denote by $\delta(u, v)$ the minimum weight of a path from u to v. If v is not reachable from u, we put $\delta(u, v) = +\infty$. Finally, if there is a path from u to v through a negative weight cycle, we put $\delta(u, v) - \infty$. The function $\delta : V \times V \to \mathbb{R} \cup \{+\infty, -\infty\}$ is called the *distance function* on (G, l). Given a source node s in a graph G, the *single-source shortest path problem* from s is the problem of finding the minimum weight paths from s to all other nodes of G or to ascertain the existence of a negative weight cycle in G reachable from s.

Most single-source shortest path algorithms are based on the *labeling* method, which maintain a *potential* function $d : V \to \mathbb{R} \cup \{+\infty\}$, a *predecessor* function $\pi : V \to V \cup \{\text{NIL}\}$, and a *status* function $S : V \to \{\text{UNREACHED}, \text{LABELED}, \text{SCANNED}\}$. Initially, one puts $d(s) := 0$, $\pi(s) :=$ NIL, $S(s) :=$ LABELED, where s is the source node, and also puts $d(v) := +\infty$, $\pi(v) =$ NIL, and $S(v) :$ UNREACHED, for $v \in V \setminus \{s\}$ (procedure INITIALIZE). Subsequently, the potential function d is updated only by assignments of the form $d(v) := d(u) + l(u, v)$, provided that $d(v) > d(u) + l(u, v)$, in which case one also puts $\pi(v) := u$ and $S(v) :=$ LABELED. It turns out that $d(v) \geq \delta(s, v)$ always holds, for $v \in V$. Additionally, if $d(v) = \delta(s, v)$, then the predecessor function π can be used to reconstruct a shortest path from s to v backwards. The values $d(v)$ are updated within SCAN operations (see below).

PROCEDURE INITIALIZE(G, s)	PROCEDURE SCAN(G, u)
for all $v \in V$ **do**	**for all** $v \in V$ such that $(u, v) \in E$ **do**
$d(v) := +\infty$	**if** $d(v) > d(u) + l(u, v)$ **then**
$\pi(v) :=$ NIL	$d(v) := d(u) + l(u, v)$
$S(v) :=$ UNREACHED	$S(v) :=$ LABELED
$d(s) := 0$	$\pi(v) := u$
$S(s) :=$ LABELED	$S(u) :=$ SCANNED

Procedure SCAN is called on LABELED nodes until all nodes are marked either UNREACHED or SCANNED. Notice that after a SCAN operation is called on a LABELED node u, some UNREACHED or SCANNED node may become LABELED, whereas the node u becomes SCANNED.

Shortest path algorithms based on the labeling method are mainly characterized by the strategy they adopt to select the next node to be scanned from the set Q of all LABELED nodes. For instance, the Bellman-Ford-Moore algorithm maintains the set of LABELED nodes into a FIFO queue. Thus, the next node to be scanned is removed from the head of the queue, whereas a node that becomes LABELED is added to the tail. As another example, Dijkstra's algorithm applies a greedy strategy, which consists in selecting at each iteration a LABELED node $v \in Q$ with the *minimum* potential value $d(v)$. It turns out that Dijkstra's algorithm is very efficient for graphs containing no negative weight edges, but it may run in exponential time if they are present.

2 The Two-Levels-Greedy algorithm

The algorithm which we propose is a natural generalization of Dijkstra's algorithm to the case in which negative weight edges are allowed, but only outside of any cycle (*negative weight edge restriction*). It consists in a *preliminary phase* and a *scanning phase*.

Given a directed graph $G = (V, E)$, with weight function l and satisfying the above negative weight edge restriction, as a first preliminary step we compute the graph $G^{SCC}(V^{SCC}, E^{SCC})$ of the strongly connected components (s.c.c.) of G. We recall that V^{SCC} is the partition of V with respect to the relation $\sim$ over V, where $u \sim v$ holds if and only if u and v lie on a same cycle, and E^{SCC} is the collection of pairs $(\mathbf{C}_1, \mathbf{C}_2)$ such that there exists an edge $(v_1, v_2) \in E$, with $v_1 \in \mathbf{C}_1$ and $v_2 \in \mathbf{C}_2$. It turns out that the graph G^{SCC} can be computed in $\mathcal{O}(|V| + |E|)$-time (cf. [CLR90]); moreover, G^{SCC} is acyclic and, because of the negative weight edge restriction, negative weight edges can connect only nodes belonging to different components. Next, again in $\mathcal{O}(|V| + |E|)$-time (cf. [CLR90]), we compute a *topological ordering* of G^{SCC}. This is any linear ordering $<$ of V^{SCC} such that if $(\mathbf{C}_1, \mathbf{C}_2) \in E^{SCC}$ then $\mathbf{C}_1 < \mathbf{C}_2$. Finally, we complete the preliminary phase by executing the procedure INITIALIZE on the graph G with a given source s. In the scanning phase, nodes are selected to be

```
TWO-LEVELS-GREEDY(G, s)
    compute the component graph G^SCC = (V^SCC, E^SCC)
    compute a topological ordering < of G^SCC
    INITIALIZE(G,s)
    while G contains some node marked LABELED do
        let C ∈ E^SCC be the <-smallest s.c.c. containing a LABELED node
        let v be a d-smallest LABELED node in C
        SCAN(G, v)
```

Fig. 1. The TWO-LEVELS-GREEDY algorithm

processed by procedure SCAN according to the following "two-levels" greedy strategy, until there is no node which is marked LABELED (for convenience, a s.c.c. $\mathbf{C}$ containing a LABELED node is said to be LABELED as well):

- firstly, the LABELED s.c.c. $\mathbf{C} \in E^{SCC}$ which is smallest with respect to the topological ordering $<$ is selected;
- secondly, a LABELED node v in $\mathbf{C}$ with minimal potential $d(v)$ is selected to be scanned.

We name the resulting algorithm TWO-LEVELS-GREEDY (TLG, for short). Its pseudo-code is shown in Fig. 2. It may be shown that under the above restriction on negative weight edges the TLG algorithm computes correctly all shortest-paths from a given source s in G. Moreover, it turns out that the TLG algorithm retains the same asymptotic time complexity as Dijkstra's algorithm. In addition, when the input graph is acyclic, its complexity is linear in the size of the graph.

3 Experimental results

A significant amount of experimental testing on shortest path algorithms has been carried out over the years, see for instance [Ber93], [CG99], [CGR96], and [MCN91]. To this purpose, several classes of graphs have been introduced. We present below experimental results relative to two extreme classes of graphs for the TLG algorithm: RAND-LEN, a class of strongly connected and dense random graphs, and ACYC-P2N, a class of acyclic random graphs with a variable fraction of negative edges (see [CGR96] for more details). All algorithms have been implemented in the **C** programming language and have been tested on a PC with AMD Athlon processor of 1.19 GHz. For each test, we computed the running time in CPU milliseconds (in bold) and the average number of scan operations per node. Maintaining the same style of [CGR96], each entry is the average of five runs of the code on problem instances produced with the same generator parameters, except for the pseudo-random generator seed. For each problem, the two best results have been underlined.

The TLG algorithm has been tested against the following algorithms: the

Bellman-Ford-Moore algorithm (BFM) and one of its variants (BFP), which implements the *parent-checking* heuristic introduced in [CGR96]; Dijkstra's algorithm implemented with bucket heaps (DIKB), as proposed by Dial [Dia69]; two incremental algorithms due to Pape (PAPE) [Pap74] and Pallottino TWOQ) [Pal84]; the threshold algorithm (THRESH) due to Glover *et. al.* [GGK84]; two topological sorting algorithms (GOR and GOR1) due to Goldberg and Radzik [GR93]. The priority queue of the TLG algorithm has been implemented with bucket heaps.

The first table presents experimental results on the RAND-LEN class of graphs, where each graph is constructed by first creating a Hamiltonian cycle and then adding edges with distinct end points. In our experiments we set to 1 the weight of the edges on the Hamiltonian cycle whereas the weights of the remaining edges have been randomly selected in the interval $[L, U]$, using a uniform distribution.

$[L, U]$	BFM	BFP	DIKB	PAPE	TWOQ	THRESH	GOR	GOR1	TLG
$[1, 1]$	**215**	**216**	**218**	**214**	**224**	**346**	**347**	**917**	**330**
	1.00	1.00	1.00	1.00	1.00	1.00	1.61	4.48	1.00
$[0, 10]$	**671**	**601**	**265**	**506**	**509**	**367**	**779**	**1124**	**360**
	3.17	2.67	1.00	2.85	2.84	1.01	4.36	5.94	1.00
$[0, 10^2]$	**1534**	**1354**	**293**	**1171**	**1229**	**705**	**1376**	**1400**	**390**
	7.40	6.14	1.00	8.07	8.05	1.78	8.93	8.21	1.00
$[0, 10^4]$	**3867**	**3463**	**339**	**3360**	**3693**	**2738**	**2573**	**2101**	**468**
	19.04	16.74	1.00	27.31	26.07	9.16	19.48	14.15	1.00
$[0, 10^8]$	**5564**	**5219**	**352**	**5500**	**5584**	**4096**	**3527**	**1520**	**478**
	29.21	26.76	1.00	47.60	41.89	16.03	27.96	12.26	1.00

The second table shows experimental results obtained on the class ACYC-P2N of acyclic random graphs. In a graph of this class all edge weights are selected uniformly from the interval $[L, U]$, where the values of $L < 0$ and $U > 0$ determine the expected fraction $f = -L/(U - L)$ of negative weight edges.

f (%)	BFM	BFP	DIKB	PAPE	TWOQ	THRESH	GOR	GOR1	TLG
0	**52**	**52**	**44**	**42**	**44**	**38**	**76**	**40**	**80**
	1.76	1.66	1.00	1.83	1.83	1.01	2.61	2.00	1.00
10	**72**	**64**	**44**	**56**	**62**	**42**	**84**	**40**	**240**
	2.43	2.21	1.14	2.64	2.64	1.29	3.01	2.00	1.00
20	**167**	**146**	**134**	**134**	**140**	**123**	**125**	**40**	**250**
	8.41	6.78	7.28	10.92	10.53	7.06	6.29	2.00	1.00
30	**731**	**542**	**771**	**516**	**488**	**619**	**195**	**40**	**270**
	40.9	28.75	41.31	69.27	59.69	40.28	12.49	2.00	1.00
40	**4702**	**3402**	**5666**	**5161**	**3304**	**4707**	**351**	**40**	**285**
	288.7	179.65	325.83	940.89	641.19	339.58	22.81	2.00	1.00
50	**32692**	**19496**	**43372**	**67320**	**29774**	**37842**	**485**	**40**	**300**
	2171.6	1056.25	2405.31	12088.96	9510.17	2862.66	30.81	2.00	1.00
60	-	**39761**	-	-	**53357**	**83537**	**587**	**38**	**320**
	-	2149.88	-	-	20124.87	5918.05	33.52	2.00	1.00
100	-	**44185**	-	-	**25967**	-	**40**	**35**	**386**
	-	2349.75	-	-	10819.75	-	2.00	2.00	1.00

Concerning the running time, it turns out that the TLG algorithm achieves in both cases very good results and often it is very close to the best performances. Concerning the number of scan operations performed by the algorithms, it turns out that the TLG algorithm always obtains the best results.

References

[Bel58] Bellman, R. E.: On a routing problem. In: *Quarterly Applied Mathematics* (1958), Nr. 16, S. 87–90

[Ber93] Bertsekas, D. P.: A simple and fast label correcting algorithm for shortest paths. In: *Networks* 23 (1993), S. 703–709

[CG99] Cherkassky, B. V. ; Goldberg, A. V.: Negative-cycle detection algorithms. In: *Mathematical Programming* 85 (1999), Nr. series A, S. 277–311

[CGR96] Cherkassky, B. V. ; Goldberg, A. V. ; Radzik, T.: Shortest paths algorithms: theory and experimental evaluation. In: *Mathematical Programming* 73 (1996), Nr. series A, S. 129–174

[CLR90] Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L.: Introduction to Algorithms. In: *MIT Press, Cambridge, MA* (1990)

[Dia69] Dial, R. B.: Algorithm 360: Shortest Path Forest with Topological Ordering. In: *Comm. ACM* 12 (1969), S. 632–633

[Dij59] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Matematik* 1 (1959), S. 269–271

[For56] Ford, L. R.: Network flow theory. In: *Rand Corporation Report* (1956), S. P–293

[FT87] Fredman, M. L. ; Tarjan, R. E.: Fibonacci heaps and Their Uses In Improved Network Optimization Algorithms. In: *J. Assoc. Comput. Mach.* 34 (1987), S. 596–615

[GGK84] Glover, F. ; Glover, R. ; Klingman, D.: Computational Study of an Improved Shortest Path Algorithm. In: *Networks* 14 (1984), S. 25–37

[GR93] Goldberg, A. V. ; Radzik, T.: A Heuristic Improvement of the Bellman-Ford Algorithm. In: *Applied Math. Let.* 6 (1993), S. 3–6

[MCN91] Mondou, J.-F. ; Crainic, T. G. ; Nguyen, S.: Shortest path algorithms: a computational study with the C programming language. In: *Computers and Operations Research* 18 (1991), S. 767–786

[Moo59] Moore, E. F.: The shortest path through a maze. In: *In Proceedings of the International Symposium on the Theory of Switching, Harvard University Press* (1959), S. 285–292

[Pal84] Pallottino, S.: Shortest-path methods: complexity, interrelations and new propositions. In: *Networks* 14 (1984), S. 257–267

[Pap74] Pape, U.: Implementation and efficiency of Moore algorithms for the shortest root problem. In: *Mathematical Programming* 7 (1974), S. 212–222