Journal of Discrete Algorithms 42 (2017) 35–44

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Hybrid Bellman–Ford–Dijkstra algorithm

Yefim Dinitz *, Rotem Itzhak

Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel

ARTICLE INFO

Article history:
Received 31 December 2012
Received in revised form 30 October 2016
Accepted 4 January 2017
Available online 7 January 2017

Keywords:
Algorithm
Graph
Shortest path
Negative edge

ABSTRACT

We consider the single-source shortest paths problem in a digraph with negative edge costs allowed. A hybrid of the Bellman–Ford and Dijkstra algorithms is suggested, improving the running time bound of Bellman–Ford for graphs with a sparse distribution of negative cost edges. The algorithm iterates Dijkstra several times without re-initializing the tentative value $d(v)$ at vertices. At most $k+2$ iterations solve the problem, if for any vertex reachable from the source, there exists a shortest path to it with at most k *negative cost* edges.
In addition, a new, straightforward proof is suggested that the Bellman–Ford algorithm produces a shortest paths tree.

1. Introduction

Several basic graph algorithms are widely known from the 1950s–1960s. Although those algorithms have being massively taught at all CS departments all over the world since then, it turns out that new aspects of them can still be revealed. We present two such results on shortest paths algorithms: a new, combined algorithm and a new correctness proof for building the shortest paths tree.

In what follows, we say "graph" meaning a digraph with edge costs. The input graph is denoted either by $G = (V, E, c)$, or by $G = (V, E, c, s)$ if the source vertex s is fixed. An edge or cycle of a negative cost are called "negative". When relating to classic results, we usually do not provide references; the reader can refer to any textbook on algorithms, e.g., [5]. In this paper, the main goal is theoretic: finding algorithms with better *worst case* running time bounds.

A special kind of innovation is hybrid algorithms: a combination of known algorithms for either solving a new problem or decreasing the time bound for an old one. For example, Dijkstra's algorithm solves the single-source *shortest* paths problem. The PERT chart algorithm finds the early time-schedule in a DAG (directed acyclic graph), whose nodes are events and the edges represent the precedence relation between them, via finding *longest* paths in it [15] (see also [5, Chapter 24.2]). The algorithm of [10,9] combines the PERT and Dijkstra algorithms for finding the early time-schedule in a *general* graph with precedence relations of AND or OR type at nodes; the PERT chart and the single-source shortest paths problems are boundary cases of this problem. Surprisingly, though these algorithms seem quite different, the hybrid algorithm combining them is natural, and its running time is the sum of running times of PERT and Dijkstra.[1]

This paper presents a new, hybrid algorithm for finding shortest paths from a source s in a graph G with general edge costs. It combines Bellman–Ford and Dijkstra algorithms, and will be called Bellman–Ford–Dijkstra (BFD). Recall that both original algorithms traverse the graph edges, executing the update (called also "relaxation" or "relabeling") of the tentative

* Corresponding author.
E-mail addresses: dinitz@cs.bgu.ac.il (Y. Dinitz), rotem.itzhak@gmail.com (R. Itzhak).

[1] A similar algorithm, formulated in the language of some grammar problem, was suggested in [14]. No attention was paid there to its hybrid nature.

http://dx.doi.org/10.1016/j.jda.2017.01.001
1570-8667/

distance function d on vertices. Dijkstra works for the non-negative edge costs case. It is a search type algorithm: it makes just a single traversal on all vertices and edges reachable from s, in a greedy order.

For graphs with negative edges, the Bellman–Ford algorithm is used. The basic Bellman–Ford works in rounds, each being a simple loop of relaxations on the graph edges, in any order. It finds the shortest path costs to the vertices and a shortest paths tree in at most $r+1$ *rounds, where r is the minimal integer such that for any vertex reachable from s, there exists a* shortest *path from s to it containing at most r edges* (in other words, r is the minimal possible depth of the shortest paths tree). If the input graph contains a negative cycle reachable from s, shortest paths to some vertices do not exist, and Bellman–Ford detects that in $|V|$ rounds. Otherwise, r as above is at most $|V|-1$. Since a single round cost is $O(|E|)$, the implied running time bound is $O(|V||E|)$. Notice that it is higher by an order of magnitude than the Dijkstra time bound $O(|E|+|V|\log|V|)$.[2]

Our goal is decreasing the (worst case) round number bound of Bellman–Ford. We achieve that at BFD by running Dijkstra at each Bellman–Ford round, without re-initializing values of d at vertices. This is a legal implementation of Bellman–Ford, since Dijkstra is just a *smart loop* of relaxations. We show that this hybrid works, despite the common knowledge: "Dijkstra's algorithm cannot handle graphs with negative edge costs".[3] Our bound for the number of BFD rounds is $k+2$, *where k is the minimal integer such that for any vertex reachable from s, there exists a* shortest *path from s to it containing at most k* ***negative*** *edges.* This is a substantial improvement over the Bellman–Ford bound for graphs with a sparse dispersion of negative edges. (It should be mentioned that the running time of a BFD round slightly increases from $O(|E|)$ to $O(|E|+|V|\log|V|)$.)

That is, the time bound of BFD improves upon that of Bellman–Ford in the in-between cases when negative edges exist, but are sparsely dispersed in the graph. A motivation of such input graphs comes naturally from classic, seemingly purely non-negative problem settings, where there are a few *special edges* in the network. For example, consider the navigation problem of finding a shortest route in a road map. Suppose that a driver chooses a few objects of interest, such as a beautiful view or a good restaurant, and assigns a route cost reduction to visiting each of them. Then, the updated road map can acquire a few negative edges.

It should be mentioned that there is a wide experimental study on *practical* shortest paths algorithms, whose running time for reasonable benchmarks is drastically lower than that defined by the known worst case bounds. For example, see [1,3] and references therein. In [3, Section 5.3] a variant of the Bellman–Ford algorithm is mentioned, where at each round the edges are traversed in the order of the values of function d at their initial vertices, which is the same order as that of BFD. However, that variant was rejected there from consideration, for practical running time reasons.

The correctness proof of BFD is a generalization of a widely known correctness proof for Bellman–Ford. In an auxiliary statement (Lemma 3.2), we analyze the *general behavior of Dijkstra algorithm in graphs with negative edges*. Hence, BFD with its proof may become an instructive supplement to a university course in algorithms.

Remark. After becoming acquainted with the Bellman–Ford–Dijkstra algorithm, Allan Borodin noted that BFD *iterates* the *greedy* Dijkstra algorithm in order to extend its applicability. Generalizing the situation, he asked what can we *prove* for a general iterated greedy algorithm? For greedy algorithms, some general lower bounds were proved in [2,6]. For iterated greedy algorithms, he conjectured that even the analysis of the two iterations case would be difficult.[4]

The paper is composed as follows. Section 2 describes the Bellman–Ford–Dijkstra algorithm. Section 3 analyzes BFD. In Section 4, we discuss the robustness of BFD to some known implementations of BF. Section 5 describes and analyzes versions of BFD adjusted to some variants of the Dijkstra's algorithm implementing its rounds. In Section 6, we show the tightness of the bounds developed for BFD, and discuss a speeding up idea.

In Appendix A, we suggest an improvement of the classic analysis of relaxation-based algorithms. A new, straightforward proof is presented that any such algorithm running on any graph produces a **shortest paths tree**. The simplification is achieved by considering the set of back-pointers *only* from the vertices, for which the tentative distance is already equal to the true one. It is shown directly that the shortest paths tree propagates together with the true distances.

2. Hybrid Bellman–Ford–Dijkstra algorithm

Recall the basic Bellman–Ford (BF) and Dijkstra algorithms, working on a directed weighted single-terminal graph $G=(V,E,c,s)$. Recall that $d(v)$ denotes the tentative distance from s to v, and $\pi(v)$ denotes the vertex that gave to $d(v)$ its current value via edge $(\pi(v),v)$.

[2] With the additional assumption that the edge lengths are integers bounded below by $-N\le -2$, Goldberg [11] suggests an algorithm with the $O(\log N\sqrt{|V|}|E|)$ time bound, for this problem.

[3] Johnson in [12] suggested a natural generalization of Dijkstra's algorithm for a graph with negative edge costs, which however was shown not to run in polynomial time.

[4] Personal communication.