# An Improved FPTAS for Restricted Shortest Path

Funda Ergun *   Rakesh Sinha †   Lisa Zhang‡

April 4, 2002

**Abstract**

Given a graph with a cost and a delay on each edge, Restricted Shortest Path (RSP) aims to find a min-cost $s$-$t$ path subject to an end-to-end delay constraint. The problem is NP-hard. In this note we present an FPTAS with an improved running time of $O(mn/\varepsilon)$ for acyclic graphs, where $m$ and $n$ denote the number of edges and nodes in the graph. Our algorithm uses a scaling and rounding technique similar to [3]. The novelty of our algorithm lies in its "adaptivity". During each iteration of our algorithm the approximation parameters are fine-tuned according to the quality of the current solution so that the running time is kept low while progress is guaranteed at each iteration. Our result improves those in [3, 4, 5].

**Key words:** Approximation algorithm, combinatorial problem.

## 1 Introduction

The Restricted Shortest Path problem (RSP) is defined as follows. Let $G$ be a graph with $n$ vertices and $m$ edges. Each edge $ij$ has an associated positive integral cost $c_{ij}$ and positive integral delay $d_{ij}$. The cost (resp. delay) of a path is defined as the summation of the costs (resp. delay) along all of its edges. We would like to find the minimum cost $s$-$t$ path in $G$ such that the total delay along this path does not exceed a given bound $D$.

RSP, besides being of theoretical interest, is also an abstraction of various "path provisioning" problems arising in Quality of Service (QoS) routing in high-speed networks. Given the QoS requirements of an application and the resource availability of the network, the goal is the identification of a feasible routing path satisfying these requirements while optimizing certain other parameters such as cost. In this context the nodes may be IP routers or intelligent optical switches running a protocol similar to Generalized Multiprotocol Label Switching (GMPLS).

RSP is known to be NP-hard [1]. In this note, we present a *fully polynomial time approximation scheme* (FPTAS), which allows a trade-off between the goodness of the approximation and the running time. Let $OPT$ denote the cost of the minimum cost path satisfying the delay constraint. We say that an algorithm gives a $(1+\varepsilon)$-approximation of the optimal solution if it finds a path of cost at most $(1+\varepsilon)OPT$ and of delay at most $D$. For any given parameter $\varepsilon > 0$, our FPTAS gives a $(1+\varepsilon)$-approximation for acyclic graphs in time $O(mn/\varepsilon)$. This is an improvement over Hassin [3], Phillips [4] and Lorenz and Raz [5]. Recently, a similar problem was discussed in [2], where an algorithm for finding a path of cost at most $OPT$ and of delay at most $(1+\varepsilon)D$ was given.

*NEC Research and Case Western Reserve University, afe@eecs.cwru.edu. Part of this work was done while the author was visiting DIMACS, New Brunswick, NJ.

†AT&T Labs, sinha@research.att.com.

‡Bell Laboratories, ylz@research.bell-labs.com.

## 2 An Exact Solution

To better understand the approximation, let us first study a dynamic programming formulation of the problem that yields an exact solution in pseudo-polynomial time. Even though RSP aims to find the minimum cost path subject to a delay constraint, the following dynamic program computes minimum delay for a given cost at each node. More precisely, let $g_j(c)$ be the minimum delay for an $s$-$j$ path of total cost at most $c$. Since the minimum delay $s$-$j$ path goes through some intermediate node $i$ (where $ij$ is an edge), we compute $g_j(c)$ by minimizing over all possible intermediate nodes $i$ (Figure 1). Since for each value of $c$ every edge is examined once, we have,

**Lemma 1** *The running time of EXACT is* $O(OPT \cdot m)$.

$EXACT(c, d)$
COMMENT: returns min-cost path with delay constraint $D$

initialization
  for all $c \geq 0$,  set $g_s(c) = 0$
  for all $j \neq s$,  set $g_j(0) = \infty$
for $c = 1, 2, \ldots$
  for all $j \neq s$,  set $g_j(c) = \min_{\text{edge } ij: c_{ij} \leq c}\{g_j(c-1), \min\{g_i(c - c_{ij}) + d_{ij}\}\}$
  if $g_t(c) \leq D$
    output $OPT = c$ and its corresponding path, exit.

Figure 1: An exact solution for $RSP$.

We remark that the dynamic programming approach in EXACT applies as long as the graph is acyclic or has positive integral costs. Hence, the values of $g_j(c)$ is computed based on $g_{j'}(c')$ where $c'$ is strictly smaller than $c$.

## 3 Approximation Algorithms and Analysis

The framework of our FPTAS is similar to [3] and is based on a technique called *rounding-and-scaling* [6]. The exact solution above has a pseudo-polynomial complexity which is proportional to $OPT$. However, if all the costs are "scaled" down enough, then the scaled optimum becomes small enough that the scaled version of the problem can be solved optimally in polynomial time. The solution is then "rounded" back to the original cost values with some bounded error. There is a trade-off, determined by the scaling factor, between the running time and the rounding error bound. A small scaling factor results in large scaled-down costs and consequently a long running time with a small error. On the other hand, a large scaling factor results in small scaled-down costs and consequently a short running time with larger error.

The trick here is to start off with a range $[LB, UB]$ of possible values for $OPT$, and then successively narrow down this range. This is achieved by using an approximate test procedure to check how a value $V$ inside the range $[LB, UB]$ compares with $OPT$. Each iteration of the test narrows down the valid range for $OPT$; when the upper and lower bounds are within a constant factor of each other, we can efficiently use a dynamic programming like procedure to obtain a $(1+\varepsilon)$-approximation. We now list the steps in this algorithm.

**Step 1.** Begin with a rough upper bound $UB$ and a rough lower bound $LB$ on $OPT$.

**Step 2.** Repeatedly apply an approximate test procedure to narrow the gap between $UB$ and $LB$ until $UB/LB \leq 2$.

**Step 3.** Obtain a $(1+\varepsilon)$-approximation given $UB/LB \leq 2$.

## 3.1 Earlier Approximations

We first describe Steps 1-3 by Hassin in [3] and by Raz and Lorenz in [5] before presenting our techniques. We do not include the proofs of Lemmas 2, 3 and 4, since the first two are implied in [3] and the last one in [5].

For Step 1 Hassin sets the initial lower bound $LB$ to 1 and the initial upper bound $UB$ to $nC$ where $C$ is the maximum edge cost. The test procedure used in Step 2 is due to Warburton [7]. (See Figure 2.) The essence of this test procedure is scaling and rounding. $TEST(V,\delta)$ scales down the cost on each edge by a factor of roughly $V\delta/n$. If a path with delay at most $D$ and scaled cost at most $\lceil n/\delta \rceil$ exists, then $TEST$ returns $OPT \leq (1+\delta)V$. Otherwise, $TEST$ returns $OPT > V$. In each iteration of Step 2, Hassin sets $\delta = \varepsilon$ and $V = \sqrt{UB \cdot LB}$, the geometric mean of the current upper and lower bounds. It takes $O(\lg\lg(nC))$ iterations to find $UB$ and $LB$ that are within a factor two of each other.

$TEST(V,\delta)$<br>
COMMENT: checks if $OPT > V$ or $OPT \leq V(1+\delta)$

initialization<br>
    for all $ij$, set $\hat{c}_{ij} = \lfloor \frac{c_{ij}}{V\delta/n} \rfloor$<br>
    for all $c \geq 0$, set $g_s(c) = 0$<br>
    for all $j \neq s$, set $g_j(0) = \infty$<br>
for $c = 1, 2, \ldots, \lfloor n/\delta \rfloor$<br>
    for all $j \neq s$, set $g_j(c) = \min_{\text{edge } ij: \hat{c}_{ij} \leq c} \{g_j(c-1), \min_i\{g_i(c-\hat{c}_{ij}) + d_{ij}\}\}$<br>
    if $g_t(c) \leq D$<br>
        output $OPT \leq V(1+\delta)$, exit<br>
output $OPT > V$

Figure 2: A $(1+\delta)$-approximation test procedure.

We remark that the scaled costs in the above TEST procedure may be zero. Therefore, the dynamic programming approach of computing $g_j(c)$ fails unless the graph is acyclic. (If the graph is acyclic the nodes can be processed in a topologically sorted order.) In order to present the successive improvements on the running time of RSP in a consistent manner, we present the algorithm for acyclic graphs. We revisit the case of cyclic graphs in Section 3.3 and show how to modify our algorithm to work with a subset (but not all) of graphs that contain cycles.

**Lemma 2** *$TEST(V,\delta)$ runs in $O(mn/\delta)$ time for $\delta \leq n$.*

For Step 3 Hassin invokes the $EXACT$ procedure using scaled costs $\hat{c}_{ij} = \lfloor \frac{c_{ij}}{LB\varepsilon/n} \rfloor$ for all edges $ij$. $OPT \leq 2LB$ implies that the scaled down optimum is $O(\frac{n}{\varepsilon})$. Substituting this in the expression for the running time of $EXACT$ (Lemma 1), we conclude

**Lemma 3** *If $UB$ and $LB$ are within a constant factor of each other, then $EXACT(\lfloor \frac{c}{LB\varepsilon/n} \rfloor, d)$ returns a path of cost at most $(1+\varepsilon)OPT$ in time $O(mn/\varepsilon)$.*

Combining Lemmas 2 and 3, Hassin obtains an FPTAS for RSP with a running time $O(\frac{mn}{\varepsilon} \cdot \lg\lg(nC))$.

Raz and Lorenz improve Step 1 by finding tighter initial bounds as stated in Lemma 4 below. Furthermore, by using $\delta = 1$ in Step 2 they bound the running time per iteration by $O(mn)$ and the number of iterations by $O(\lg\lg n)$. Hence, the overall running time of their FPTAS is $O(mn \cdot \lg\lg n + mn/\varepsilon)$.

**Lemma 4** *An initial $UB$ and $LB$ where $UB \leq nLB$ can be found in $O(n\lg^2 n + m\lg n)$ time.*

## 3.2 Speeding up the Approximation

Following Lemma 4, we begin with $UB$ and $LB$ where $UB \leq nLB$. We show how to reduce the gap such that $UB \leq 2LB$ in $O(mn)$ time. Combining Lemma 3 we obtain an FPTAS with running time $O(mn/\varepsilon)$.

Our key idea is to choose $\delta$ to be a function of $UB/LB$ when applying the $TEST$ procedure repeatedly in Step 2. The running time and quality of approximation depend on the value of $\delta$. The intuition is that initially when $UB$ and $LB$ are far apart, we can afford to choose a large $\delta$ and apply a very coarse approximation: the test procedure will be much faster, and it will still narrow down the gap between $UB$ and $LB$ significantly. As $UB$ and $LB$ get closer, we choose smaller $\delta$ and apply finer approximations. More precisely, at each iteration, we let,

$$\delta = \sqrt{UB/LB} - 1. \tag{1}$$

We refer to our algorithm as ADAPT and present it in Figure 3.

$ADAPT(\varepsilon)$
COMMENT: returns $(1+\varepsilon)$-approximation

1 initialize $UB$ and $LB$ s.t. $UB \leq nLB$
2 while $UB > 2LB$
3     set $\delta = \sqrt{UB/LB} - 1$
4     set $V = \sqrt{UB \cdot LB/(1+\delta)}$
5     if $TEST(V, \delta)$ outputs $OPT \geq V$
6         set $LB = V$
7     else set $UB = (1+\delta)V$
8 obtain $(1+\varepsilon)$-approximation by invoking $EXACT(\lfloor \frac{c}{LB\varepsilon/n} \rfloor, d)$

Figure 3: ADAPT, our FPTAS.

**Lemma 5** *If initially $UB \leq nLB$, then Step 2 takes $O(mn)$ time in our algorithm.*

**Proof:** Let $UB_i$, $LB_i$, $V_i$ and $\delta_i$ be the parameters used in the $i$th application of $TEST$. At the end of this application, we know that either $OPT \geq V_i$ or that $OPT \leq (1+\delta_i)V_i$. In the former case we set the new lower bound $LB_{i+1} = V_i$ and the new upper bound $UB_{i+1} = UB_i$, and therefore $UB_{i+1}/LB_{i+1} = UB_i/V_i$. In the latter case we set the new upper bound $UB_{i+1} = V_i(1 + \delta_i)$ and $LB_{i+1} = LB_i$, and therefore $UB_{i+1}/LB_{i+1} = V_i(1 + \delta_i)/LB_i$. By the definitions of $V_i$ and $\delta_i$, the following holds for both of the above cases.

$$UB_{i+1}/LB_{i+1} = (UB_i/LB_i)^{3/4}. \tag{2}$$

Let $k$ be the number of applications of $TEST$. By Lemma 2, the total time required to narrow down the upper and lower bound to within a factor two is $\sum_{1\leq i\leq k} O(mn/\delta_i)$. It therefore suffices to show that $\sum_{1\leq i\leq k} 1/\delta_i = O(1)$. The definition of $\delta_i$ implies $1/\delta_i = 1/(\sqrt{UB_i/LB_i} - 1)$. Since $UB_i > 2LB_i$ for $i \leq k$, we have $\sqrt{LB_i/UB_i} \leq 1/\delta_i \leq (2 + \sqrt{2})\sqrt{LB_i/UB_i}$. Hence, $\sum_{1\leq i\leq k} 1/\delta_i = O(\sum_{1\leq i\leq k} \sqrt{LB_i/UB_i})$. We also have,

$$\begin{aligned}
\sum_{1\leq i\leq k} \sqrt{LB_i/UB_i} &= \sum_{0\leq j<k} (LB_k/UB_k)^{\frac{1}{2}\cdot\left(\frac{4}{3}\right)^j} \\
&\leq \sum_{0\leq j<k} 2^{-\frac{1}{2}\cdot\left(\frac{4}{3}\right)^j} \\
&\leq 2^{-\frac{1}{2}} \sum_{0\leq j<k} p^j \\
&\leq 2^{-\frac{1}{2}}/(1-p) \\
&\leq 6.5,
\end{aligned}$$

where $p = 2^{-\frac{1}{6}}$. The first equality follows from repeatedly applying Equation (2). The first inequality follows from the fact that $UB_k > 2LB_k$. The second inequality holds since $2^{-\frac{1}{2}\cdot\left(\frac{4}{3}\right)^{j+1}} \leq p \cdot 2^{-\frac{1}{2}\cdot\left(\frac{4}{3}\right)^j}$ for $j \geq 0$. □

Given the running times for Step 2 and Step 3, combining Lemmas 4, 5 and 3, we obtain,

**Theorem 6** *ADAPT, our FPTAS for RSP, has running time $O(mn/\varepsilon)$.*

## 3.3 Graphs with Cycles

Recall that the scaled costs $\hat{c}_{ij}$ in the procedure $TEST(V, \delta)$ may be zero and hence the dynamic programming approach fails for graphs with cycles. One obvious fix is to define scaled cost by $\hat{c}_{ij} = \lfloor \frac{c_{ij}}{V\delta/n} \rfloor + 1$ instead of $\hat{c}_{ij} = \lfloor \frac{c_{ij}}{V\delta/n} \rfloor$. (Modifying Line 2 of Figure 2.) Unfortunately, such a fix does not lead to a $(1 + \delta)$-approximation test procedure with desired running time of $O(mn/\delta)$ if $\delta > 1$.

It is not obvious whether there are any easy ways of generalizing our algorithm to work for all cyclic graphs. However, a simple modification works for a large class of cyclic graphs, the only exception being graphs that contain a cycle such that *all* the edges in the cycle has zero modified cost.

We propose the following modification to $TEST(V, \delta)$. We compute $g_t(c)$ for all $c \leq \lfloor n/\delta \rfloor$ in one large graph which we call $H$. For a graph $G$ whose edges $ij$ have cost $c_{ij}$ and delay $d_{ij}$, we create a graph $H$ as follows. For each node $i$ in $G$, we create $\lfloor n/\delta \rfloor$ nodes in $H$ and label them $(i, c)$ for $0 \leq c \leq \lfloor n/\delta \rfloor$. For each edge $ij$ in $G$, we add an edge between nodes $(i, c)$ and $(i, c + \hat{c}_{ij})$ and let this edge have delay $d_{ij}$. The new graph $H$ has at most $m\lfloor n/\delta \rfloor$ edges, and at most $n\lfloor n/\delta \rfloor$ nodes. We modify our $TEST$ algorithm to work on this new graph $H$; note that for the algorithm to work properly, $H$ needs to be acyclic.

The modified $TEST$ procedure is given in Figure 4. Note that, since $H$ is acyclic, Line 6 takes $O(mn/\epsilon)$ time. Thus, *MODIFIED-TEST* still takes total time $O(mn/\epsilon)$ and the overall algorithm remains unchanged.

We now explore for which graphs $G$ the corresponding $H$ are acyclic. Note that along any path in $H$ the costs are nondecreasing; i.e., if node $(i, c_2)$ follows $(j, c_1)$, then $c_2 \geq c_1$. Thus, if $H$ has a cycle, the costs of the nodes in the cycle are the same. More precisely, the nodes around a cycle of size $k$ are of the form $(i_1, c_1), \ldots (i_k, c_k)$, where $c_1 = \ldots = c_k$. This happens only when all the edges in the corresponding cycle in $G$ have zero scaled down cost. So even if $G$ has cycles, $H$ will be acyclic as long as one or more edges in every cycle of $G$ have a positive scaled cost.

$MODIFIED - TEST(V, \delta)$
COMMENT: checks if $OPT > V$ or $OPT \leq V(1 + \delta)$

1 initialization
2 for all $ij$, set $\hat{c}_{ij} = \lfloor \frac{c_{ij}}{V\delta/n} \rfloor$
3 create graph $H$
4 for all $i$, create nodes $(i, c)$ for $0 \leq c \leq \lfloor n/\delta \rfloor$
5 for all $ij$, create edges between nodes $(i, c)$ and $(j, c + \hat{c}_{ij})$ for $0 \leq c \leq \lfloor n/\delta \rfloor - \hat{c}_{ij}$
let these edges have delay $d_{ij}$
6 find min delay in $H$ from $(s, 0)$ to $(t, c)$ for $0 \leq c \leq \lfloor n/\delta \rfloor$
7 if any delay $\leq \lfloor n/\delta \rfloor$
8 output $OPT \leq V(1 + \delta)$
9 else
10 output $OPT > V$

Figure 4: A modified $(1 + \delta)$-approximation test procedure.

# References


[1] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[2] A. Goel, K.G. Ramakrishnan, D. Kataria, and D. Logothetis. Efficient computation of delay-sensitive routes from one source to all destinations. In *Proceedings INFOCOM* 2001.

[3] R. Hassin. Approximation schemes for the restricted shortest path problems. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[4] C. Phillips. The network inhibition problem. In *Proceedings 25th Annual ACM Symposium on the Theory of Computing*, pages 776–785, San Diego, CA, May 1993.

[5] D. Raz and D. Lorenz. Simple efficient approximation scheme for the restricted shortest path problem. Technical Report 10009674-991214-04TM, Bell Labs, 1999.

[6] S. Sahni. Algorithms for scheduling independent tasks. *Journal of ACM*, 23:116–127, 1976.

[7] A. Warburton. Approximation of pareto optima in multiple-objective, shortest path problems. *Operations Research*, 35:70–79, 1987.