CS273 Foundations of Parallel and Network Computing **Spring 2003**

Lecture 8: Feb 19, 2003

Lecturer: Satish Rao *Scribe: Hoeteck Wee*

Disclaimer: *These notes*[1] *have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

8.1 Outline

We started discussing path selection on arbitrary networks in the previous lecture by introducing the linear program for the problem. In this lecture, we will show how to use randomized rounding to obtain a solution with a strong approximation guarantee. In addition, we shall examine a simpler algorithm that does not use the power of linear programming.

8.2 Path Selection via Linear Programming

8.2.1 Linear Program 1

Recall the path selection problem:

> Given a graph $G = (V, E)$ on n vertices and k pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$, find k paths connecting s_i to t_i for all i, such that the maximum number of paths using any edge (that is, the *congestion*) is minimized.

We denote the optimal congestion by C_{opt}.

Intuitively, the path selection problem requires that we find k paths connecting each pair of vertices s_i, t_i so that no edge is used too much. This definition gives rise naturally to the following linear program:

> For each path p in G, let $f(p)$ be the indicator variable indicating whether the path is chosen, so that it assumes the value 1 if p is chosen, and 0 otherwise. We want to find assignments to $f(p)$ so as to:

$$
\begin{array}{lll}
\textbf{minimize} & C & \\
\textbf{subject to} & \displaystyle\sum_{\text{paths } p \text{ connecting } s_i, t_i} f(p) = 1 & \text{for all } i \\
 & \displaystyle\sum_{\text{paths } p \text{ that use } e} f(p) \leq C & \text{for all edges } e \\
 & 0 \leq f(p) \leq 1 & \text{for all paths } p \text{ in } G
\end{array}
$$

[1]Some of the text and diagrams in this set of notes were contributed by Kunal Talwar, who scribed a similar lecture in 2001.

The first constraint specifies that there is exactly one path joining each pair of vertices (s_i, t_i), and the second constraint specifies that the congestion on each edge $e \in E$ is at most C. The third constraint represents a relaxation of the requirement that $f(p)$ be $0-1$ variables. It is therefore not immediately clear how we can derive an integer solution from solving the linear program.

Another problem with this approach is that there are exponentially many paths p in the graph G, and therefore the linear program uses exponentially many variables, one for each path p. However, there is a method due to Khachiyan, called the ellipsoid method that can solve a linear program with exponentially many variables and polynomially many constraints (under certain conditions, of course). The method works in this case, and we can therefore solve the above linear program in polynomial time. Nonetheless, let us consider a different linear program that uses only polynomially many variables.

8.2.2 Linear Program 2

We can think of the path selection problem as routing one unit of flow from s_i to t_i for each i while minimizing the maximum flow on the edges. To cast this as a linear program, let us define indicator variables $f^i(e)$ for $i = 1, 2, \ldots, k$ and all edges $e \in E$ such that $f^i(e)$ is 1 if edge e lies on the selected path from s_i to t_i, and 0 otherwise.

We can then enforce the constraints on the variables as local conditions of flow conservation, wherein the flow leaving each vertex equals the flow entering the vertex, except for the sources s_i, which have a flow of 1 leaving and 0 entering, and for the sinks t_i, which have a flow of 1 entering and 0 leaving. This yields only a polynomial number of constraints.

Now, we have:

$$
\begin{array}{ll}
\textbf{minimize} & C \\
\textbf{subject to} & \displaystyle\sum_{i=1}^{k} f^i(e) \leq C \qquad \text{for all } e \in E \\
& \displaystyle\sum_{e=(u,v)\in E} f^i(e) - \sum_{e=(v,w)\in E} f^i(e) = \begin{cases} 0, & \text{if } v \in V \setminus \{s_i, t_i\}; \\ -1, & \text{if } v = s_i; \\ 1, & \text{if } v = t_i. \end{cases} \qquad \text{for all } v \in V,\ i = 1, 2, \ldots, k \\
& 0 \leq f^i(e) \leq 1 \qquad \text{for all } e \in E,\ i = 1, 2, \ldots, k
\end{array}
$$

Again, we relax the requirement that $f^i(e)$ be $0-1$ variables. In this setting, we can interpret $f^i(e)$ as the flow of commodity i (which starts at s_i and ends at t_i) along edge e.

8.2.3 Equivalence of the LPs

We may derive an optimal solution to the first linear program from an optimal solution to the second. Let $f^i(e)$ be an optimum for the second LP. Consider a particular i and look at the directed graph of edges with non-zero $f^i(e)$. While this graph can have cycles, we get rid of all cycles to get a directed acyclic graph. Now find a path p from s_i to t_i greedily. Assign $f^i(p)$ to be the minimum $f^i(e)$ along this path and deduct this amount from each of the edges on this path (see figure 8.1). Recur. It is important here that we have conservation of flow at each vertex.

This computation is efficient because each step deletes at least one edge, so we terminate in at most $|E|$ steps. Moreover we find a solution that has only polynomially number of nonzero variables. It is also clear that we have found an optimal assignment of $f^i(p)$'s, since the congestion for this choice of paths is equal to that for the optimal solution to the second linear program.

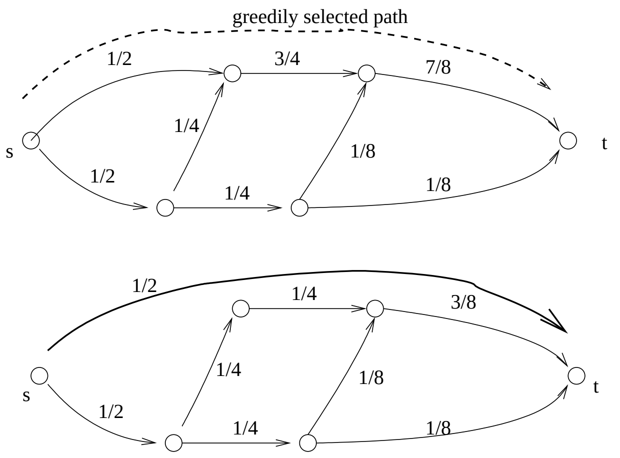

Figure 8.1: One step for deriving $f(p)$'s from $f(e)$'s. The top diagram shows the edges of a graph labelled with $f^i(e)$ for some fixed i, and we have marked on it the path p that we have greedily selected. Here, we have $f(p) = 1/2$.

8.2.4 Randomized Rounding

In this section, we address the problem that solving the two linear programs can yield fractional optimal solutions instead of integer solutions. To do that, we introduce a procedure known as *randomized rounding* for obtaining an integer solution that approximates the optimal solution to the linear programs.

First, we can set up and solve the second linear program, and from there derive a fractional solution f_{LP}, C_{LP} to the first linear program, so that for each i, we have a small (polynomial-sized) set of paths P_i, such that $f^i_{LP}(p) > 0$ for all paths $p \in P_i$ and $\sum_{p \in P_i} f^i_{LP}(p) = 1$. We can interpret the quantities $f^i_{LP}(p)$ as probabilities, and for each i, pick a path $p \in P_i$ with probability $f^i_{LP}(p)$. This gives us exactly k paths $p_1, \ldots, p_k$, where p_i is a path from s_i to t_i.

Recall that the optimal solution to the path selection problem also yields a feasible solution to the linear program, so C_{LP} is a lower bound on the optimal congestion for path selection problem C_{opt}, that is, $C_{LP} \leq C_{opt}$.

Analyzing congestion on the paths selected using randomized rounding uses same trick as for that analyzing congestion on the butterfly network. Fix an edge e in E, and let X_i be indicator variable that commodity i chooses this edge, $i = 1, 2, \ldots, k$. Then, we have:

$$\mathrm{E}[\,X_i] = \sum_{\text{paths } p \text{ in } P_i \text{ that use edge } e} f^i_{LP}(p)$$

$$\mathrm{E}[\,\text{congestion at } e] = \sum_{i=1}^{k} \mathrm{E}[\,X_i] = \sum_{\text{paths } p \text{ that use edge } e} f_{LP}(p) \leq C_{LP} \leq C_{opt}$$

The paths $p_1, \ldots, p_k$ are chosen independently, so the variables $X_1, \ldots, X_k$ are completely independent. As such, we may apply Chernoff bounds to analyze the tail probabilities to obtain

$$\Pr[\text{congestion at } e > C_{opt} + c_1\sqrt{C_{opt}\log n} + c_2 \log n] < 1/poly(n)$$

As a general rule of the thumb, we use Chernoff bound to bound sums of $0-1$ variables when the summands have unequal means, and a direct counting argument when they have equal means.

We may now apply a union bound over all edges $e \in E$ to deduce that the maximum congestion is $C_{opt} + O(\sqrt{C_{opt} \log n} + \log n)$ with high probability.

We can then repeat randomized running $O(\log n)$ times, compute the congestion in each repetition and select the collection of paths that yields the smallest congestion. This gives us an efficient randomized approximation algorithm for the path selection problem, where the relationship between the congestion C_{max} in the solution obtained and the optimal congestion C_{opt} is as follows (with high probability):

$$\begin{array}{ll} C_{opt} \approx O(1): & C_{max} \approx O(\log n) \\ C_{opt} \approx \log n: & C_{max} \approx O(C_{opt}) \end{array}$$

Randomized Rounding of linear programs to get solutions for integer programs has wide use, but is not a magic bullet. For example, it is easy to setup an integer linear program for 3-SAT. But the fractional relaxation admits a trivial solution. So, solving the linear program and rounding is probably not helpful.

8.3 A Simpler Deterministic Algorithm for Path Selection

In this section, we present a simpler combinatorial algorithm for path selection that does not use linear programming, and has the advantage of being deterministic. Again, this algorithm only returns a solution with a strong approximation guarantee.

8.3.1 A First Attempt

We can view the path selection problem as an optimization problem, wherein we have to connect every pair of vertices (s_i, t_i) while optimizing the usage of edges (our resource). Again, we can see this as routing one unit of flow between each pair of vertices (s_i, t_i).

Suppose we take the collection $\mathcal{C}_{sp}$ of shortest paths between s_i, t_i, and let $d(s_i, t_i)$ denote the length of the shortest path between s_i and t_i.

Let us count the number of edges used in any selection of paths between all pairs (s_i, t_i) (with multiplicities, so that an edge used by 2 different paths is counted twice, etc). Clearly, we must use at least $\sum_i d(s_i, t_i)$ edges, because we need $d(s_i, t_i)$ to connect s_i and t_i. It follows that the average congestion is at least $\frac{\sum_i d(s_i,t_i)}{|E|}$, and therefore $\mathcal{C}_{sp}$ achieves the minimum possible average congestion.

Furthermore, we know for the optimal selection of paths, that each edge in E is used at most C_{opt} times. Hence, the number of edges used here (with multiplicities) is at most $|E| \cdot C_{opt}$, and thus $|E| \cdot C_{opt} \geq \sum_i d(s_i, t_i)$, from which we have:

$$C_{opt} \geq \frac{\sum_i d(s_i, t_i)}{|E|} \tag{8.1}$$

On the other hand, $\mathcal{C}_{sp}$ can yield a choice of paths that has arbitrarily high congestion relative to the optimal selection of paths, as shown in figure 8.2.

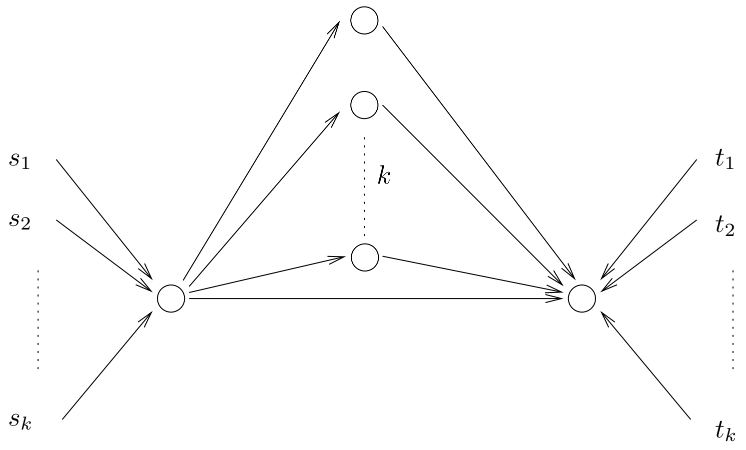

Figure 8.2: An example where the shortest path heuristic fails badly. In particular, selecting all pairs of shortest paths yields a congestion of k, whereas the optimal path selection yields a congestion of 1.

8.3.2 The Algorithm SimplePathSelect

The problem with choosing the collection of shortest paths in figure 8.2 we have used the center edge too often. In particular, after we have picked the shortest path between s_1 and t_1, we should avoid choosing a path between s_2 and t_2, etc that uses the center edge.

We can fix this problem by making an edge that is heavily congested longer. In particular, we assign a distance function to edges depending on their congestion as follows:

$$d(e) = \alpha^{f(e)} \qquad \text{for all edges } e$$

where $f(e)$ is the current flow on edge e, and $\alpha > 1$ is a fixed parameter. Note that $d(\cdot)$ is extremely sensitive to the current flow.

Now, pick the shortest paths under the $d(\cdot)$ distance metric. Once the paths are chosen, the distances (and thus the shortest paths) change, so we will need to update the distances and reroute the flows. The algorithm stated as it is may not converge, but we may modify it as follows:

Algorithm SimplePathSelect
1. for every pair (s_i, t_i) **do**
1.1 find the shortest $s_i - t_i$ path p_i and route along it
1.2 update the length function d
2. while there is some (s_i, t_i) such that $d(p_i) \geq 2\alpha\, d(s_i, t_i)$
(where p_i is the current path connecting s_i to t_i, and $d(s_i, t_i)$
is the length of the shortest path p'_i under the updated $d(\cdot)$) **do**
2.1 reroute commodity i along p'_i and set p_i equal p'_i
2.2 update the distance function d
3. return $p_1, \ldots, p_k$.

Upon termination, we have: for all i, $d(p_i) < 2\alpha\, d(s_i, t_i)$. It is not immediately clear that this algorithm will terminate (and in a polynomial number of rounds); we will prove this at the end of the lecture.

8.3.3 How Good is SimplePathSelect?

Let C_{max} denote the congestion on the paths $p_1, \ldots, p_k$ selected by the SimplePathSelect when the algorithm terminates, and f the corresponding flow. To analyze how far C_{max} is from C_{opt}, we would first need a lower bound on C_{opt}:

Claim 8.1 *Give any distance function $d : E \to \mathbb{R}^+$,*

$$C_{opt} \geq \frac{\sum_{i=1}^{k} d(s_i, t_i)}{\sum_{e \in E} d(e)}$$

Proof: The proof is similar to that for equation (8.1). We can think of distance function as defining a series of pipes, where for each edge e, we have a pipe of length $d(e)$ and capacity C_{opt}. Therefore, the pipes can carry a total flow of at most $C_{opt} \cdot \sum_e d(e)$. On the other hand, if we pick the optimal flow, we use at least $d(s_i, t_i)$ of volume to route from s_i to t_i, so the total volume of flow is at least $\sum_i d(s_i, t_i)$. Thus, $C_{opt} \cdot \sum d(e) \geq \sum_i d(s_i, t_i)$, and the claim follows.

To see this more formally, consider the optimal selection of paths $p_1^*, \ldots, p_k^*$ in the original path selection problem (wherein each edge has distance 1), so the congestion is at most C_{opt} on every edge. Let f_{opt} denote the corresponding flow.

$$\begin{aligned}
C_{opt} \sum_{e \in E} d(e) &\geq \sum_{e \in E} f_{opt}(e) d(e) \\
&= \sum_i \sum_{e \in p_i^*} f_{opt}(p_i^*) d(e) \\
&= \sum_i \sum_{e \in p_i^*} d(e) \qquad \text{since } f_{opt}(p_i^*) = 1 \text{ for all } i \\
&= \sum_i d(p_i^*) \\
&\geq \sum_i d(s_i, t_i)
\end{aligned}$$

■

Applying Claim 8.1 to the distance function $d(\cdot)$ that we get when SimplePathSelect terminates, we have:

$$\begin{aligned}
2\alpha\, C_{opt} &\geq \frac{2\alpha \sum_i d(s_i, t_i)}{\sum_e d(e)} \\
&\geq \frac{\sum_i d(p_i)}{\sum_e d(e)} \qquad \text{using } d(p_i) < 2\alpha\, d(s_i, t_i) \\
&= \frac{\sum_i f(p_i) d(p_i)}{\sum d(e)} \qquad \text{again, } f(p_i) = 1 \text{ for all } i \\
&= \frac{\sum_e f(e) d(e)}{\sum_e d(e)}
\end{aligned}$$

Recall that $d(e) = \alpha^{f(e)}$. Let n_ℓ be the number of edges with flow ℓ, $0 \leq \ell \leq C_{opt}$. Then,

$$2\alpha\, C_{opt} \geq \frac{\sum_e \alpha^{f(e)} f(e)}{\sum_e \alpha^{f(e)}} = \frac{\sum_{\ell=0}^{C_{max}} \alpha^\ell n_\ell \ell}{\sum_{\ell=0}^{C_{max}} \alpha^\ell n_\ell} \tag{8.2}$$

Observe that the sums in both the numerator and the denominator on the right-hand-side are dominated by the terms corresponding to large values of ℓ, that is, the edges with high flows. In particular, if every edge has flow C_{max}, then the right-hand-side equals C_{max}. Now, to obtain a lower bound for the right-hand-side, let us separate the edges into two groups: those with flow less than $\ell' = C_{max} - 2\log_\alpha |E|$, and those with flow at least ℓ'.

For the edges with low flow, we have:

$$\begin{aligned}
\sum_{\ell \le \ell'} \alpha^\ell n_\ell &\le \alpha^{\ell'} \sum_{\ell \le \ell'} n_\ell \\
&= \frac{\alpha^{C_{max}}}{|E|^2} \sum_{\ell \le \ell'} n_\ell && \text{where } \alpha^{\ell'} = \tfrac{\alpha^{C_{max}}}{|E|^2} \text{ follows from our choice of } \ell' \\
&\le \frac{\alpha^{C_{max}}}{|E|} && \text{because } \textstyle\sum_{\ell \le \ell'} n_\ell \le \sum_{\ell=0}^{C_{max}} n_\ell = |E| \\
&\le \frac{1}{|E|} \sum_{\ell > \ell'} \alpha^\ell n_\ell && \text{because at least one edge has flow } C_{max}\text{, that is, } n_{C_{max}} \ge 1
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
\sum_{\ell=0}^{C_{max}} \alpha^\ell n_\ell \ell &\ge \sum_{\ell > \ell'} \alpha^\ell n_\ell \ell \ge \ell' \sum_{\ell > \ell'} \alpha^\ell n_\ell \\
\sum_{\ell=0}^{C_{max}} \alpha^\ell n_\ell &= \sum_{\ell \le \ell'} \alpha^\ell n_\ell + \sum_{\ell > \ell'} \alpha^\ell n_\ell \le \left(1 + \frac{1}{|E|}\right) \sum_{\ell > \ell'} \alpha^\ell n_\ell
\end{aligned}$$

This allows us to rewrite (8.2) as:

$$2\alpha\, C_{opt} \ge \frac{\ell' \sum_{\ell > \ell'} \alpha^\ell n_\ell}{\left(1 + \frac{1}{|E|}\right) \sum_{\ell > \ell'} \alpha^\ell n_\ell} = \frac{\ell'}{\left(1 + \frac{1}{|E|}\right)} = \frac{C_{max} - 2\log_\alpha |E|}{\left(1 + \frac{1}{|E|}\right)} \tag{8.3}$$

Note that the lower bound on the right-hand-side is quite comparable to C_{max}, which is an estimate we made based on the assumption that all edges have high flow equal to C_{max}. This supports our intuition that only the edges (our resource) with high congestion matter, which shows up in the way the analysis is done too.

Rearranging the terms in (8.3) and substituting $|E| \ge 1$ yields an upper bound on C_{max}:

$$C_{max} \le 2\alpha \left(1 + \frac{1}{|E|}\right) C_{opt} + 2\log_\alpha |E| \le 4\alpha C_{opt} + 2\log_\alpha |E|$$

Looking at different range of parameters C_{opt}, we have:

$C_{opt} \approx O(1)$:	$C_{max} \approx O(\log \lvert E \rvert)$	set $\alpha = O(1)$
$C_{opt} \approx O(1)$:	$C_{max} \approx O\left(\frac{\log \lvert E \rvert}{\log\log \lvert E \rvert}\right)$	set $\alpha = \theta\left(\frac{\log \lvert E \rvert}{\log\log \lvert E \rvert}\right)$
$C_{opt} \approx \log \lvert E \rvert$:	$C_{max} \approx O(C_{opt})$	set $\alpha = O(1)$
$C_{opt} \gg \log \lvert E \rvert$:	$C_{max} \approx 4\alpha C_{opt}$	for any $\alpha > 1$

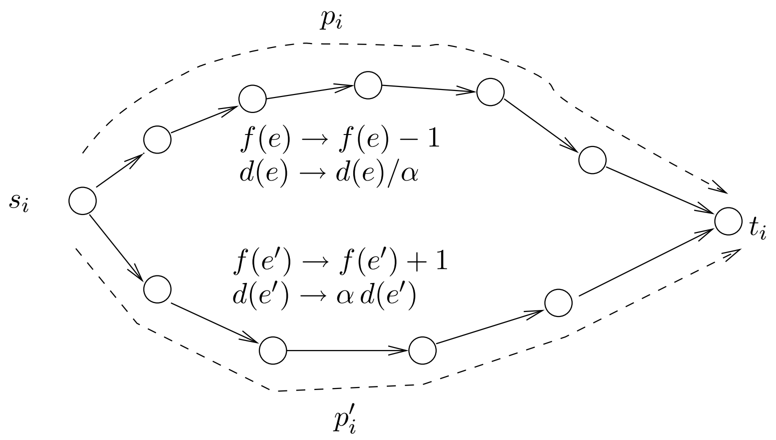

Figure 8.3: Convergence of SimplePathSelect: Path p_i is replaced by the shortest path p'_i in an iteration of step 2, and suppose $e \in p_i, e' \notin p_i$ and $e' \in p'_i, e' \notin p_i$. The expressions show how d and f are affected after the updates.

8.3.4 Relationship to LP Algorithm

The function $d(\cdot)$ comes from the LP duality and corresponds to variables in the dual to the second linear program, and we are essentially solving the dual linear program in SimplePathSelect. In particular, Claim 8.1 follows from the weak LP duality theorem. The approach used here for converting LP algorithms to combinatorial algorithms (that is, using resource based on cost function, and re-using resource based on cost function) works for many linear programs.

An advantage of the combinatorial algorithm SimplePathSelect is that it yields paths with approximately the same congestion as randomized rounding, without using any randomness, that is, it is a deterministic algorithm. In addition, SimplePathSelect can be modified slightly and with a tighter analysis, can be shown to yield a $(1+\epsilon)$ approximation to the optimal solution with a $poly(n, |E|, 1/\epsilon)$ running time. In this regard, the LP algorithm is more powerful, because it can be modified to yield a $(1+\epsilon)$ approximation with a $poly(n, |E|, \log(1/\epsilon))$ running time.

8.3.5 Convergence of SimplePathSelect

We now show that SimplePathSelect does converge.

Claim 8.2 *In each iteration of step 2 of* SimplePathSelect, $\sum_{e \in E} d(e)$ *strictly decreases.*

Proof: Suppose that in some iteration of step 2, we have to reroute commodity i, wherein the path p_i from s_i to t_i is replaced by the shortest path p'_i (see figure 8.3). Let the new flow be f', and the updated distance function be d'. Then we have,

$$f'(e) = \begin{cases} f(e), & \text{if } e \notin p_i \text{ and } e \notin p'_i; \\ f(e)+1, & \text{if } e \notin p_i \text{ and } e \in p'_i; \\ f(e)-1, & \text{if } e \in p_i \text{ and } e \notin p'_i; \\ f(e), & \text{if } e \in p_i \text{ and } e \in p'_i. \end{cases}$$

and therefore

$$d'(e) \quad = \quad \begin{cases} d(e), & \text{if } e \notin p_i \text{ and } e \notin p_i'; \\ \alpha\, d(e), & \text{if } e \notin p_i \text{ and } e \in p_i'; \\ \frac{1}{\alpha} d(e), & \text{if } e \in p_i \text{ and } e \notin p_i'; \\ d(e), & \text{if } e \in p_i \text{ and } e \in p_i'. \end{cases}$$

Thus

$$\begin{aligned}
& \sum_{e \in E} d'(e) - \sum_{e \in E} d(e) \\
= \quad & \sum_{e \in p_i' \setminus p_i} (d'(e) - d(e)) + \sum_{e \in p_i \setminus p_i'} (d'(e) - d(e)) \\
= \quad & (\alpha - 1) \sum_{e \in p_i' \setminus p_i} d(e) - \left(1 - \frac{1}{\alpha}\right) \sum_{e \in p_i \setminus p_i'} d(e) \\
\leq \quad & (\alpha - 1) \sum_{e \in p_i'} d(e) - \left(1 - \frac{1}{\alpha}\right) \sum_{e \in p_i} d(e) \quad \text{because } (\alpha - 1) d(e) \geq \left(1 - \tfrac{1}{\alpha}\right) d(e) = \tfrac{\alpha - 1}{\alpha} d(e) \text{ all } e \in p_i \cup p_i' \\
\leq \quad & (\alpha - 1) d(p_i') - \left(1 - \frac{1}{\alpha}\right) \cdot 2\alpha\, d(p_i') \qquad \text{since } d(p_i) \geq 2\alpha\, d(s_i, t_i) = 2\alpha\, d(p_i') \\
= \quad & -(\alpha - 1) d(p_i') < 0
\end{aligned}$$

■

Therefore, the algorithm converges. Using a more careful analysis, it can be shown that $\sum_e d(e)$ decreases by at least a factor of $(1 - \frac{1}{|E|})$ in each round, and hence the algorithm terminates in polynomially many rounds.