Constructing Optimal Shortcuts in Directed Weighted Paths

R. Klein* M. Krug† E. Langetepe* D. T. Lee‡ D. Wagner†

Abstract

We show that the optimum shortcut, with respect to routing cost, in a directed path with weighted vertices can be found in linear time. To this end, we construct in linear time the lower envelope of an arrangement of pseudo-lines whose order at infinity is given. The algorithm can also be applied to star-shaped directed networks.

Keywords: Arrangements, computational geometry, lower envelope, network optimization, pseudo-line, routing cost, shortcut edge.

1 Introduction

Transportation networks can be modelled by graphs in a number of different ways. Quite frequently, vertices represent cities, and edges stand for highway or railroad connections between them. In the *routing cost* model, the overall quality of a network is measured by the sum of the lenghts of all shortest paths between all pairs of vertices; see Wu and Chao [6]. Here the length of a path equals the sum of the lengths of its edges.

We are interested in augmenting a given network, by adding an extra edge to it in such a way that its routing cost is reduced as much as possible. In our model each edge is of unit length, reflecting a situation where long distance traffic is fast, while changing between highways or railways requires time consuming inner-city travel.

This augmenting problem has been studied by Farshi et al. [2] with respect to the dilation (or: stretch factor, spanning ratio) of a network. Their problem differs considerably from ours. For example, the geometric path P over n vertices shown in Figure 1 is of dilation $2/\epsilon$, attained by vertices v_1 and v_3, while its routing cost

$$2 \sum_{1\le i<j\le n} (j-i) = \frac{(n-1)n(n+1)}{6}$$

does not depend on the geometric embedding of P. With one extra edge available, we would minimize the dilation of P by inserting it between v_1 and v_3, whereas the routing cost of P is minimized by adding an edge connecting $v := v_{n/5}$ to $v' := v_{4n/5}$; the proof of optimality requires lenghty calculations. Not only the paths between vertices to the left of v and to the right of v' become shorter; vertices between v and v' also benefit from the shortcut edge, as Figure 1 indicates.

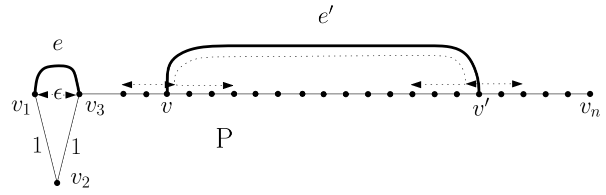

Figure 1: Inserting edge e minimizes the dilation, while e' minimizes the routing cost of path P.

In this note we consider the following problem. We are given a path $P = (v_1, v_2, \ldots, v_n)$ all of whose edges (v_i, v_{i+1}) are directed from left to right. Each vertex v_i is assigned a weight $w_i > 0$, reflecting the number of residents of city v_i. We want to decrease the routing cost

$$r := \sum_{1\le i<j\le n} w_i(j-i)w_j$$

of P by adding one more edge (v_k, v_l) directed from v_k to v_l, choosing $k < l$ such that the decrease in routing cost is as large as possible; compare Figure 2.

As compared to the undirected case shown in Figure 1, the situation has become simpler in that only vertex pairs on opposite sides of the shortcut edge can benefit from the extra edge. Thus, the decrease in routing cost effected by adding a directed edge from v_k to v_l equals

$$\rho(k,l) := \left(\sum_{1\le i\le k} w_i\right)(l-k-1)\left(\sum_{l\le i\le n} w_i\right), \qquad (1)$$

because $l-k$ edges between v_k and v_l can be avoided by using edge (v_k, v_l).

On the other hand, introducing weights has made the minimization problem non-trivial. In fact, the decrease function $\rho(k,l)$ can have multiple local maxima. (An example is given by the path with weights $(90, 10, 100, 1, 100, 1, 1, 1, 100, 1, 100, 10, 90)$ for which the shortcut edges (v_3, v_{11}) and (v_5, v_9) are locally optimal.) Quite obviously, the optimum shortcut edge could be determined by inspecting the $O(n^2)$ many candidates (v_k, v_l) where $1 \le k < l \le n$.

*Department of Computer Science I, University of Bonn
†Karlsruhe Institute of Technology (KIT), Institute for Theoretical Informatics
‡Institute of Information Science, Academia Sinica

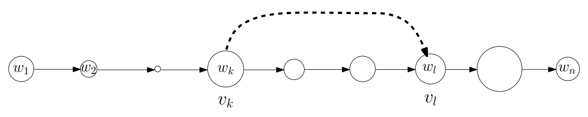

Figure 2: Weights w_i represent the population size of city v_i.

In this note, we are going to prove a sharper result.

Theorem 1 *The optimum shortcut edge for a directed, weighted path of n vertices can be determined in time $O(n)$.*

2 Reduction to Pseudo-Line Arrangements

Let P denote a directed path of n vertices $v_1, \dots v_n$ with positive weights $w_1, \dots, w_n$. Considering Formula 1 we define

$$A_k := \sum_{i=1}^{k} w_i \quad \text{and} \quad B_l := \sum_{i=l}^{n} w_i$$

for $k = 1, \dots, n$. Moreover, let

$$f_l(k) := A_{\lfloor k \rfloor}(l - k - 1)B_l$$

be a function of a real variable k. In our optimization problem, only values $k < l$ are meaningful, but there is no harm in ignoring this constraint. Function $f_l(k)$ is piecewise linear, with discontinuities at integer values of k. Here, $f_l(k) = \rho(k, l)$ holds.

Lemma 2 *Let $1 \leq l_1 < l_2 \leq n$ be fixed.*
(i) There exists at most one real value $k \in [1, n]$ where $f_{l_1}(k) = f_{l_2}(k)$ holds. If so, we have $f_{l_1}(k) < f_{l_2}(k)$ to the right of k.
(ii) If no such value exist, $f_{l_1}(k) < f_{l_2}(k)$ holds over the whole interval $[1, n]$,

Proof. Let us consider

$$f_{l_2}(k) - f_{l_1}(k) = A_{\lfloor k \rfloor} \cdot (\ (l_2 - 1)B_{l_2} - (l_1 - 1)B_{l_1} + k(B_{l_1} - B_{l_2})\ ). \quad (2)$$

Because of $l_1 < l_2$ we have $B_{l_1} > B_{l_2}$. Moreover, $A_{\lfloor k \rfloor} \leq A_{\lfloor k' \rfloor}$ holds if $k < k'$. Thus, Expression 2 is strictly monotonically increasing in k, which proves Claim (i). If Expression 2 never attains the value 0, then the order relation between $f_{l_1}(k)$ and $f_{l_2}(k)$ is the same all over $[1, n]$. If we set $k := l_2 - 1 \in [1, n]$ we obtain $f_{l_1}(k) < 0 = f_{l_2}(k)$, which completes the proof of Claim (ii). □

The set of graphs of functions $f_l(k)$, where $1 \leq l \leq n$, is called a family of *pseudo-lines* over the interval $[1, n]$ because any two of them have at most one point of intersection, just as proper lines would. Arrangements of pseudo-lines have been extensively studied; see Goodman [4], for example. We can solve our optimization problem by constructing the *upper envelope* of this pseudo-line arrangement, that is, the graph G of the maximum function

$$f(k) := \max\{f_l(k); 1 \leq l \leq n\}. \quad (3)$$

Each function $f_l(k)$ contributes at most one segment to G. Indeed, let us assume that some $f_l(k)$ contributed two segments to G. Then a segment of some $f_{l'}(k)$, where $l' \neq l$, must occur in between. But for this to happen, $f_l(k)$ and $f_{l'}(k)$ must intersect twice—in contradiction to Lemma 2.

This is a special, and in fact the most simple, case of a Davenport-Schinzel sequence. In general, if any two of n function graphs over some interval intersect at most s times, their envelope is of complexity $O(\lambda_s(n))$, with a non-trivial, slightly super-linear function λ_s; see the monograph by Sharir and Agarwal [5]. There is a simple algorithm that allows the lower (or upper) envelope to be constructed in time $O(\lambda_s(n) \log n)$, by divide-and-conquer. Here one assumes that elementary operations, like computing an intersection of two functions, can be carried out in constant time. In our case, this is true. Moreover, $s = 1$ and $\lambda_1(n) = n$ hold.

These facts give us a first improvement over the trivial $O(n^2)$ algorithm mentioned in Section 1. We can construct the upper envelope, G, in time $O(n \log n)$. Then we perform one pass over G and evaluate $f(k)$ at all integer values of k; this takes time $O(n)$. If the maximum of these values is attained within a segment of $f_l(k)$ in G, then the shortcut edge from v_k to v_l yields a maximum reduction in routing cost.

3 Computing the Envelope in Linear Time

To improve on the $O(n \log n)$ upper time bound just mentioned, we will make use of the fact that the ordering of the functions $f_l(k)$ "far to the right" is known to us. Indeed, for each $l \leq n - 1$, Expression 2 yields

$$f_{l+1}(n) - f_l(n) = A_n(B_{l+1} + (n + 1 - l)w_l) > 0,$$

so that we obtain

$$f_1(n) < f_2(n) < \dots < f_n(n).$$

We prove a general result covering this situation.

Theorem 3 *Let $f_1, \dots, f_n$ form an arrangement A of pseudo-lines over interval $I = [a, b]$. If the order of values $f_i(b)$ is known, the upper envelope of A can be computed in time $O(n)$.*

Proof. We proceed from right to left and process the curves f_j in order of decreasing indices. In a stack, S, we store the part of the upper envelope that would result if no further curves existed; see Figure 3. Initially, S contains only f_n.

When processing f_j, we compute the intersection, w, of f_j with the top element, f_k, of S. If w does not exist, or lies to the left of interval $I = [a, b]$, we ignore f_j, and start processing f_{j-1}.

Otherwise, let v be the intersection of the two topmost curves in S. If w lies to the left of v, or if no v exists because the stack contains only one element, $f_k = f_n$, we push f_j on the stack, and start processing f_{j-1}. But if w lies to the right of v, we pop f_k from the stack and continue processing f_j. This pop

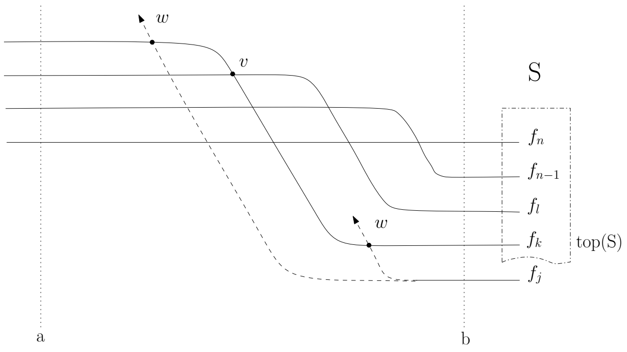

Figure 3: Constructing the upper envelope.

operation is justified because to the left of w, curve f_k is dominated by f_j and can, therefore, not contribute to the upper envelope. With this observation, the correctness of our algorithm is evident.

The linear run time bound can be shown as follows. During each pass through the loop described above, we (i) ignore f_j, or (ii) push f_j, or (iii) we pop f_k. Whenever (i) or (ii) occurs, we decrease the index j of the function currently processed; thus, (i) and (ii) happen at most n times. If there are only n push operations, there can be no more than n pop operations either, because each pop is sucessful. Thus, the loop is carried out at most $O(n)$ times, which proves the linear time bound. Upon termination, stack S contains the segments of the upper envelope, with the leftmost segment on top. □

This completes the proof of Theorem 1. Finally, we construct the upper envelope in linear time and find its maximum among all integer arguments in $[1, n]$, as explained at the end of Section 2.

Applied to strict lines there is an interesting relation to a variant of *Graham's scan* algorithm for computing the convex hull of a set of sorted points. If *Andrew's monotone chain method* [1] is translated into the dual space, we obtain the above algorithm for a set of strict lines sorted by slope.

4 Extensions

The most simple extension of a one-way path might be a star-shaped directed network as depicted in Figure 4. For a center vertex v we have s incoming path of complexity (number of nodes) m_i for $i = 1, \dots, s$ and r outgoing paths of complexity n_j for $j = 1, \dots, r$, as depicted in Figure 4. There are three possible locations for a shortut. The shortcut might be placed fully inside an incoming path, fully inside an outgoing path or it might connect an incoming path with an outgoing path. For the first two cases we can ap-

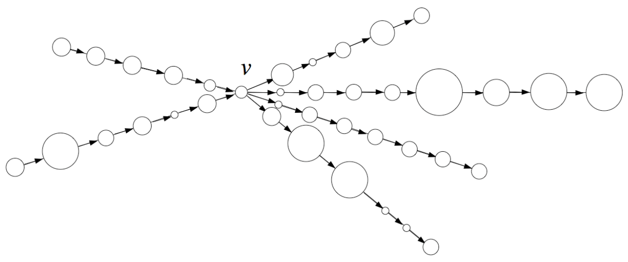

Figure 4: A directed star network with center v, 2 incoming and 4 outgoing paths of different length.

ply the same idea as presented above. We only have to sum up all weights of the incoming paths and all weights of the outgoing paths at v in order to adapt the functions $f_l(k)$ properly. Then we let our algorithm run on every path and obtain a total running time of $\sum_{i=1}^{s} m_i + \sum_{j=1}^{r} n_j$ which is optimal.

So the remaining task is to compute the best shortcut connecting an incoming with an outgoing path. In a direct way we could apply our algorithm to all $s \times r$ alternatives which results in running time $\sum_{i,j}(m_i + n_j)$, obviously this is quadratic. We can improve on this.

First, let us assume that there is only a single incoming path with complexity m. Applying the algorithm for all r outgoing paths using the same incoming path r times gives overall running time $\sum_{j=1}^{r} n_j + r \cdot m$. For a set of s incoming paths we intuitively will collect all incoming paths in a single incoming path of length $\max m_i$. Then the running time for computing the best shortcut between an incoming and an outgoing path is given by $\sum_{j=1}^{r} n_j + r \cdot \max m_i$.

The idea depends on the special nature of the functions $f_l(k)$. Within the algorithm of Section 3 we have to compute intersections in order to compute the upper envelope. In our special case we rather have computed intersection indices. Computing an intersection index means that we have to find out where $f_{l_1}(k) - f_{l_2}(k)$ changes its sign for some $1 \le k < l_1 < l_2 \le n$. From Equation (2) we obtain that $f_{l_1}(k) - f_{l_2}(k)$ for some index k is given by $A_k \cdot (\ (l_2 - 1 + k)B_{l_2} - (l_1 - 1 + k)B_{l_1}\ )$. Fortunately, the sign of this expression is independent from A_k since

A_k is always positive. Furthermore $(l_2 - 1 + k)$ and $(l_2 - 1 + k)$ are only path distances from v_{l_1} and v_{l_2} to vertex v_k, respectively. This means that knowing the exact value of the indices l_1 and l_2 and k with respect to n is not necessary for computing intersections. Therefore we can apply the above algorithm for an outgoing path (starting from the end and ending at v) without knowing the exact path length n of the path in the beginning and without knowing the exact values A_k to the left. Intersection indices are computed as path distances relative to the given position in the path. In principle we rather compute functions $f_{v_l}(k)$ instead of functions $f_l(k)$ where k is the path distance of a directed path from some vertex w to v_l. The intersection indices computed for a single outgoing path are the same for all incoming paths.

Of course, at the end we have to compute the envelopes to obtain the maximum. At this point we have to make use of the values A_k. A *collected* incoming path will simply represent the maximum values A_k for a given path distance from v. The construction of the single incoming path works as follows. Since we have to compare all incoming paths, we have to be independent from indices. Let v_k be a vertex of an incoming path. Instead of A_k we will denote the sum of weights *arriving* at v_k by A_{v_k}. For any path distance d we compute the vertex $w(d)$ of an incoming path with path distance d to v that has maximum weight $A_{w(d)}$ among all such vertices of the same path distance d from v on all incoming paths. It is easy to see that we can compute these vertices in overall running time $\sum_{i=1}^{s} m_i$. Now the *collected* incoming path is given by vertices $w(\max m_i), w(\max m_i - 1), \ldots, w(1)$ in this order with corresponding weight sums $A_{w(k)}$.

Theorem 4 *Given a star-shaped directed network with s incoming paths of complexity n_i for $i = 1, \ldots, s$ and r outgoing paths of complexity m_j for $j = 1, \ldots, r$. The optimal shortcut can be computed in $\sum_{i=1}^{s} m_i + \sum_{j=1}^{r} n_j + r \cdot \max m_i$ time.*

Finally, we consider the undirected case of a single path. As already depicted in Figure 1 there are different parts of the path that might profit from the shortcut between v and v' or v_k and v_l. A corresponding function $\phi(k, l)$ for indices $l < k$ becomes more complicated. Two vertices profit, if the path length along the shortcut is smaller than the original path length. We collect the benefit of a shortcut in the following functions due to the vertices that benefit.

First formula (from left to right as before):

$$\left(\sum_{i=1}^{k} w_i\right) \times \left(\sum_{j=l}^{n} w_j\right) \times (l - k - 1) \tag{4}$$

Second formula (from left to inner right):

$$\left(\sum_{i=1}^{k} w_i\right) \times \left(\sum_{j=\lceil \frac{l+k}{2}+1 \rceil}^{l-1} w_j \times (2j - l - k - 1)\right) \tag{5}$$

Third formula (from right to inner left):

$$\left(\sum_{i=l}^{n} w_i\right) \times \left(\sum_{j=k+1}^{\lfloor \frac{l+k}{2}-1 \rfloor} w_j \times (l + k - 2j - 1)\right) \tag{6}$$

Fourth formula (from inner right to inner left):

$$\sum_{i=\lceil \frac{l+k}{2}+1 \rceil+1}^{l-1} w_i \times \left(\sum_{j=k+1}^{i-\lceil \frac{l+k}{2}+1 \rceil+k} w_j \times (k - l + 2i - 2j - 1)\right) \tag{7}$$

Now let $\phi(k, l)$ be the sum of the function (4),(5),(6) and (7). It can be shown that one can choose weights so that two functions $\phi(k, l_1)$ and $\phi(k, l_2)$ will have more than one intersection. This means that the key idea of Theorem 1 does not apply directly to undirected paths. Note, that the best shortcut can be easily computed in $O(n^2)$ time anyway.

5 Conclusion

We have shown how to find an optimum shortcut in an oriented path with weighted vertices, by efficiently constructing upper envelopes of pseudo-lines. In case of a star-shaped directed network the algorithm can be applied also. Natural questions are if one can find efficient algorithms for more general graph classes; even the case of undirected paths is open.

Acknowledgements: We would like to thank an anonymous referee for pointing out some interrelation to a variant of *Graham's scan*; see Section 3.

References

[1] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. Inform. Process. Lett. 9(5), pp 216–219, 1979

[2] M. Farshi, P. Giannopoulos, and J. Gudmundsson. Improving the stretch factor of a geometric graph by edge augmentation. SIAM Journal on Computing, 38(1), pp. 226–240, 2008.

[3] M. Fischetti, G. Lancia, and P. Serafini. Exact Algorithms for Minimum Routing Cost Trees. Networks 39(3), pp. 161–173, 2002.

[4] J. E. Goodman. Pseudoline Arrangements. In J. E. Goodman and J. ORourke (eds.) Handbook of Comb. and Computat. Geometry, CRC Press, 1997.

[5] M. Sharir and P. Agarwal. Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, 1995.

[6] B. Ye Wu and K.-M. Chao. Spanning trees and optimization problems. Chapman & Hall/CRC, 2004.

[7] Wu, Bang Ye and Lancia. A Polynomial-Time Approximation Scheme for Minimum Routing Cost Spanning Trees. SIAM J. Comput. 29(3), pp. 761–778, 2000.