Faster All-Pairs Optimal Electric Car Routing

Dani Dorfman[*] Haim Kaplan[†] Robert E. Tarjan[‡] Mikkel Thorup[§] Uri Zwick[†]

Abstract

We present a randomized $\tilde{O}(n^{3.5})$-time algorithm for computing *optimal energetic paths* for an electric car between all pairs of vertices in an n-vertex directed graph with positive and negative *costs*, or *gains*, which are defined to be the negatives of the costs. The optimal energetic paths are finite and well-defined even if the graph contains negative-cost, or equivalently, positive-gain, cycles. This makes the problem much more challenging than standard shortest paths problems.

More specifically, for every two vertices s and t in the graph, the algorithm computes $\alpha_B(s,t)$, the maximum amount of charge the car can reach t with, if it starts at s with full battery, i.e., with charge B, where B is the capacity of the battery. The algorithm also outputs a concise description of the optimal energetic paths that achieve these values. In the presence of positive-gain cycles, optimal paths are not necessarily simple. For dense graphs, our new $\tilde{O}(n^{3.5})$ time algorithm improves on a previous $\tilde{O}(mn^2)$-time algorithm of Dorfman et al. [ESA 2023] for the problem.

The *gain* of an arc is the amount of charge added to the battery of the car when traversing the arc. The charge in the battery can never exceed the capacity B of the battery and can never be negative. An arc of positive gain may correspond, for example, to a downhill road segment, while an arc with a negative gain may correspond to an uphill segment. A positive-gain cycle, if one exists, can be used in certain cases to charge the battery to its capacity. This makes the problem more interesting and more challenging. As mentioned, optimal energetic paths are well-defined even in the presence of positive-gain cycles. Positive-gain cycles may arise when certain road segments have magnetic charging strips, or when the electric car has solar panels.

Combined with a result of Dorfman et al. [SOSA 2024], this also provides a randomized $\tilde{O}(n^{3.5})$-time algorithm for computing *minimum-cost paths* between all pairs of vertices in an n-vertex graph when the battery can be externally recharged, at varying costs, at intermediate vertices.

1 Introduction

Let $G = (V, A, c)$ be a weighted directed graph, where V is the set of vertices, $A \subseteq V \times V$ is the set of arcs, and where $c : A \to \mathbb{R}$ is a real-valued *cost* function defined on the arcs. The cost $c(uv)$ of an arc $uv \in A$ [1] is the amount of energy consumed when traversing the arc. Throughout most of this paper, it is more convenient to work with a gain function $g : A \to \mathbb{R}$ rather than a cost function. The *gain* $g(uv)$ of an arc $uv \in A$ is simply $g(uv) = -c(uv)$, i.e., the amount of energy gained by traversing the arc. The gain $g(uv)$ is negative if moving from u to v requires spending energy, or positive if energy is gained by moving from u to v.

A weighted directed graph $G = (V, A, g)$, where $g : A \to \mathbb{R}$ is a gain function, may be viewed as modeling a road network on which an electric car can roam. The electric car is assumed to have a

[*]Max Planck Institute for Informatics, Saarbrücken , Germany. Email: `dani.i.dorfman@gmail.com`.

[†]`{haimk,zwick}@tau.ac.il`. Work of Uri Zwick partially supported by grant 2854/20 of the Israeli Science Foundation. Work of Haim Kaplan partially supported by ISF grant 1595/19 and the Blavatnik family foundation.

[‡]Department of Computer Science, Princeton University. Research partially supported by a gift from Microsoft. Email: `ret@princeton.edu`.

[§]BARC, University of Copenhagen, Denmark. Research supported by the VILLUM Foundation grant no. 16582. Email: `mikkel2thorup@gmail.com`

[1]For brevity we denote an arc from u to v by uv, rather than (u, v).

battery of *capacity* B, where $B > 0$ is a parameter, i.e., it can store up to B units of energy. The *charge*, i.e., the amount of energy in the battery, can never be negative, and can never exceed the capacity of the battery. If the car is currently at vertex u with charge b in its battery, where $0 \leq b \leq B$, then it can traverse an arc $uv \in A$ if and only if $b + g(uv) \geq 0$. If this condition holds, and the car traverses the arc, then it reaches v with a charge of $\min\{b+g(uv), B\}$. The car can traverse uv if $b+g(uv) > B$, but the battery does not charge beyond its capacity of B. The car can traverse a path if and only if it can sequentially traverse its arcs. Throughout most of the paper we assume that no external charging of the battery is allowed. The battery is only charged by traversing arcs with positive gain. We may assume that $g(uv) \in [-B, B]$, for every $uv \in A$, as arcs with $g(uv) < -B$ can never be used, and can thus be removed, and gains $g(uv) > B$ can be changed to $g(uv) = B$ without changing the problem.

We consider the following two related natural questions:

1. Given two vertices $s, t \in V$, what is the *maximum final charge*, denoted $\alpha_B(s,t)$, with which the car can reach t if it starts at s with full battery, i.e., with a charge of B? If the car cannot reach t even with an initial charge of B at s, we let $\alpha_B(s,t) = -\infty$. More generally, we let $\alpha_b(s,t)$, where $0 \leq b \leq B$, be the maximum final charge with which the car can reach t if it starts at s with a charge of b.
2. Given two vertices $s, t \in V$, what is the *minimum initial charge* at s, denoted $\beta_0(s,t)$, that enables the car to reach t? If the car cannot reach t even with an initial charge of B at s, we let $\beta_0(s,t) = \infty$. More generally, we let $\beta_b(s,t)$, where $0 \leq b \leq B$, be the minimum initial charge at s required for reaching t with a charge of at least b.

It is not difficult to see, as shown in Dorfman et al. [5, Corollary 5.2], that $\beta_0(s,t) = B - \overleftarrow{\alpha}_B(t,s)$, where $\overleftarrow{\alpha}_B(t,s)$ denotes the maximum final charge at s when starting at t with full battery in the *reverse* of the graph. Thus, the problems of computing maximal final charges and minimum initial charges are computationally equivalent. (Note, however, that due to the reverse operation used, the single-source version of the maximum final charge problem becomes equivalent to the single-target version of the minimum initial charge problem.) In this paper, we only work with maximal final charges.

If all arc costs are nonnegative, i.e., all gains are nonpositive, then it is easy to see that $\beta_0(s,t) = \delta(s,t)$, and $\alpha_B(s,t) = B - \delta(s,t)$, if $\delta(s,t) \leq B$, where $\delta(s,t)$ is the standard distance from s to t with respect to the costs of the arcs. Otherwise, $\beta_0(s,t) = \infty$ and $\alpha_B(s,t) = -\infty$. When costs and gains can be both positive and negatives, the problem becomes more complicated. If there are no positive-gain cycles in the graph, the problem can be solved using fairly simple adaptations of standard shortest paths algorithms. Thus, the single-source version of the maximal final charges problem can be solved in $O(mn)$ time using an adaptation of the classical Bellman-Ford algorithm [2, 7], and the all-pairs version of the problem can be solved in $O(mn + n^2 \log n)$ time by an adaptation of the classical algorithm of Johnson [9]. For these results see, Artmeier, Haselmayr, Leucker and Sachenbacher [1], Eisner, Funke and Storandt [6], Brim and Chaloupka [3], and Dorfman, Kaplan, Tarjan and Zwick [5].

The problem becomes much harder when the graph may contain positive-gain cycles. Part of the difficulty is that optimal paths, which are still well-defined, are not necessarily simple and might have to 'hop' from one positive-gain cycle to another, until gaining enough charge to head directly to the destination. (See Lemma C.5 below.) Hélouët et al. [8] obtained a polynomial time algorithm for the decision problem of determining whether $\beta_0(s,t) \leq B$. Dorfman et al. [5] obtained an $O(mn+n^2 \log n)$-time algorithm for the single-source version of the problem, which of course implies an $O(mn^2 + n^3 \log n)$-time algorithm for the all-pairs version.

Our main result is a randomized $\tilde{O}(n^{3.5})$-time[2] algorithm for solving the all-pairs versions of the maximal final charge problem, and hence also the minimum initial charge problem, improving by a $\Theta(\sqrt{n})$ factor for sufficiently dense graphs on the $O(mn^2 + n^3 \log n)$ running time of the algorithm of

[2]The $\tilde{O}(\cdot)$ hides logarithmic factors.

Dorfman et al. [5]. To appreciate our result, we draw a parallel to standard shortest paths. On a graph with n nodes and m edges (and a suitable potential function), the single source shortest path problem can be solved in $O(m + n \log n)$ time, leading to an $O(mn + n^2 \log n) = O(n^3)$ all pairs algorithm for dense graphs. A breakthrough result by Williams [11] achieved an $O(\frac{n^3}{2^{c\sqrt{\log n}}})$ time all pairs algorithm, shaving a subpolynomial factor for dense graphs.

All the discussion so far assumed that that battery cannot be recharged at intermediate vertices. A natural variant is obtained when we assume that the battery can be charged at some of the vertices of the graph, with a cost per unit of charge that may vary from vertex to vertex. The goal then, is to find *minimum-cost paths* within all pairs of vertices in the graph. This problem was considered by Khuller, Malekian and Mestre [10] in the context of conventional, gas-operated, cars, i.e., when all arc costs are positive, and by Dorfman, Kaplan, Tarjan, Thorup and Zwick [4] in the context of electric cars, i.e., when the costs, or gains, can be both positive and negative, and where there might be positive-gain cycles. The main result of Dorfman et al. [4] is a reduction from the all-pairs minimum-cost paths problem to the all-pairs maximal final charges and minimum initial charges problems, and to the standard all-pairs shortest paths problem. Combined with the results of Dorfman et al. [5], this implies an $O(mn + n^2 \log n)$-time algorithm for the all-pairs minimum-cost paths in graphs with no positive-gain cycles. Combined with our result, we obtain a randomized $\tilde{O}(n^{3.5})$-time algorithm for the all-pairs minimum-cost paths in graphs that may contain positive-gain cycles.

To obtain the improved algorithm we need to introduce many new ideas. We next try to give a rough intuitive description of some of them, ignoring some technicalities that will be dealt with later.

Optimal energetic paths can be very long. (Their length cannot be bounded as a function of n alone. A bound must also take the arc gains and the capacity of the battery into account. For more details, see [5].) A natural idea to reduce the length of optimal energetic paths is to introduce *shortcuts*, i.e., add new arcs that correspond to possibly long paths in the graph. In the standard shortest paths problem, any path in the graph can be used to generate a shortcut, with the gain (or cost) of the arc equal to the sum of the gains of the arcs on the path. This is far from being the case for energetic paths. Consider, for example, a path xyz with $g(xy) = -1$ and $g(yz) = 1$. We cannot add a new arc xz with $g(xz) = 0$ to the graph since an electric car with an empty battery would be able to traverse the new arc xz, but not the original path xyz.

Ignoring some technicalities, we can add a shortcut corresponding to a traversable path in the graph (a path that can be traversed if we start with full battery) if the path is *ascending* or *descending*. (See Figure 1(a)-(b).) Given a path $u_0u_1 \ldots u_k$, let $a_i = \sum_{j=0}^{i-1} g(u_ju_{j+1})$, for $0 \le j \le k$, be the prefix sums of the gains along the path. We say that a path is *ascending* if $0 \le a_i \le a_k$, for every $1 \le i \le k$, and descending if $a_k \le a_i \le 0$, for every $1 \le i \le k$. If $u_0u_1 \ldots u_k$ is ascending or descending, then we are allowed to add a shortcut u_0u_k with $g(u_0u_k) = a_k$. For brevity, we refer to ascending or descending paths as *monotone*.[3]

Unfortunately, most paths are not monotone. Furthermore, subpaths of monotone paths are not necessarily monotone. There may also be very long paths that do not contain any monotone subpath. We refer to such paths as *funnels*. Examples of funnels are given in Figure 1(c)-(f).

Our algorithm constructs monotone paths and funnels and combines them to obtain new monotone paths and funnels until enough information is available to find the optimal energetic paths. The exact details, some of which are quite delicate, appear in the rest of the paper.

Another idea used by our new algorithm is *sampling*. It is well known that a random set of vertices of size $(cn \log n)/k$ is likely to hit any given path of length at least k. Taking advantage of this fact in our context is again much more complicated.

The rest of this extended abstract consists of a technical review of our algorithm and main techniques.

[3]Note that this does not imply that the prefix sums $a_1, a_2, \ldots, a_k$ form a monotone sequence.

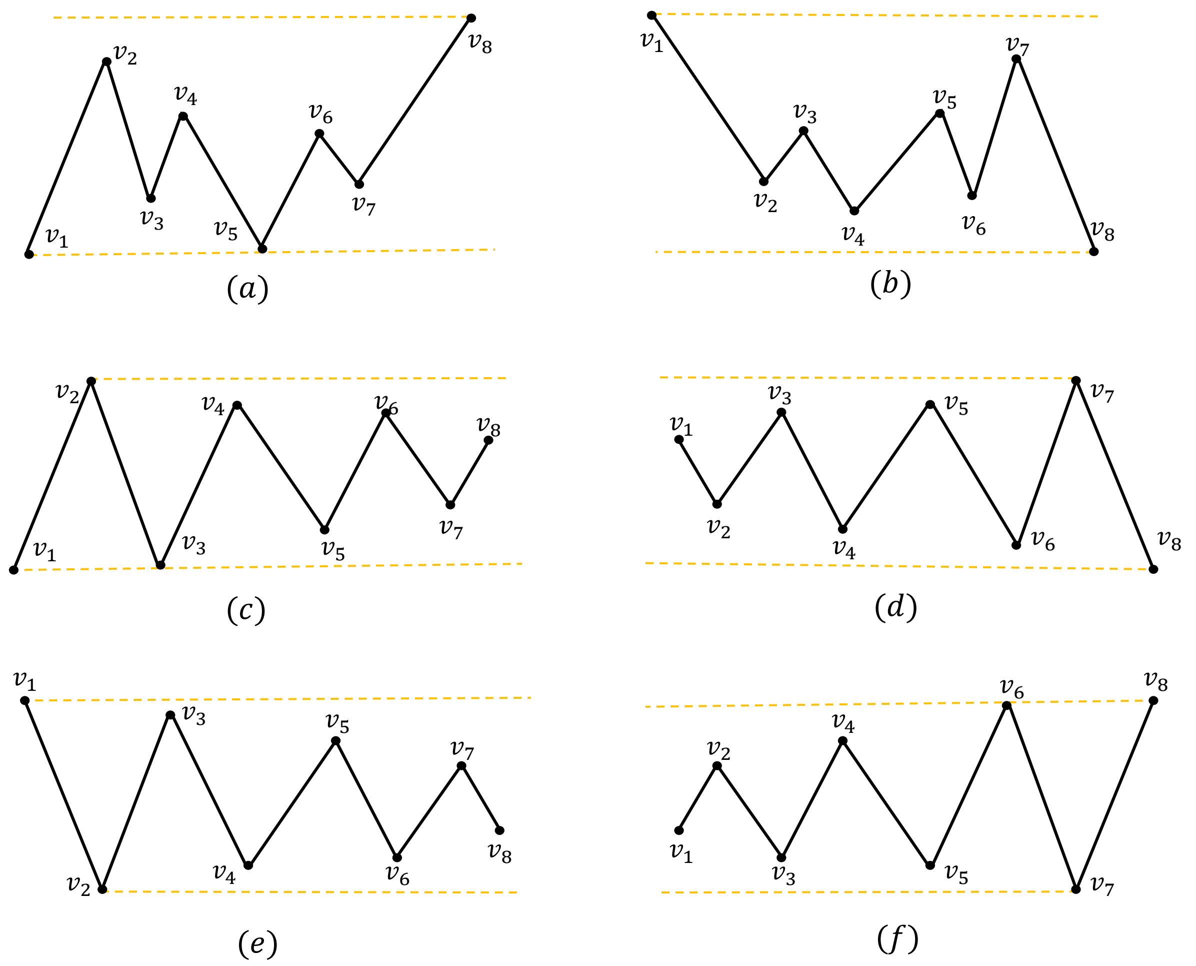


Figure 1: The graphs represent directed paths going from left to right. The vertical height of an arc e in the figure is $|g(e)|$. The vertical height of a vertex is its gain on the path (i.e. sum of arc gains). Figures (a) and (b) show an ascending path and a descending path, respectively. Figures (c)-(f) show the four possible cases for funnels. Note that in Figure (c), v_3 (which is the endpoint of the second arc of the funnel) has the same gain as v_1, this is valid.

The full version of the paper is given in the appendix.

2 Technical Review

A main tool in our algorithm is *shortcutting.* In the setting of standard shortest paths, any path $P = v_1 \dots v_k$ can be shortcutted to a single arc $v_1 v_k$ of gain $g(P) = \sum_{i=1}^{k-1} g(v_i v_{i+1})$ without affecting the lengths of the shortest paths. Unfortunately, because of the upper and lower bound constraints on the battery, this technique breaks down when applied to energetic paths. That is, by shortcutting arbitrary paths, we may change the optimal energetic paths. For example, assume $B = 10$ and let G be a graph that is composed of two paths $P_1 = v_1 v_2 v_3$ and $P_2 = u_1 u_2 u_3$, where $g(v_1 v_2) = g(u_2 u_3) = -5$ and $g(v_2 v_3) = g(u_1 u_2) = 5$. Observe that $\alpha_0(v_1, v_3) = -\infty$ and $\alpha_{10}(u_1, u_3) = 5$. On the other hand, by shortcutting the paths $v_1 v_2 v_3$ and $u_1 u_2 u_3$ (to arcs of gain 0) we will be able to reach v_3 from v_1 when starting with zero charge. Moreover by using the new 0 gain arc $u_1 u_3$ the maximum final charge at u_3 (when starting with 10 charge at u_1) becomes 10.

The above discussion encourages us to find *safe* paths that can be shortcutted without affecting the optimal energetic paths (i.e., without affecting the α values). We call these paths *monotone paths*, see Definition B.2 and Figure 1. Monotone paths are either *ascending* or *descending*. An ascending path $P = v_1 \dots v_k$ is a *traversable* path that satisfies that whenever an electric car traverses P, the car has minimum charge at v_1 and maximum charge at v_k. A traversable path $P = v_1 \dots v_k$ is a path that does not contain any subpath $v_i \dots v_j$ of gain smaller than $-B$ (this is equivalent to saying that a car that starts at v_1 with full charge can traverse P without the charge level going below zero). Similarly, a descending path $P = v_1 \dots v_k$ is a traversable path that satisfies that whenever an electric car traverses P, the car has max charge at v_1 and minimum charge at v_k (in particular, the gain of a descending path is at least $-B$). A monotone path avoids the two problems mentioned in the previous example: The charge level of an ascending path never drops below the charge level at v_1 and therefore the path $v_1v_2v_3$ from the previous example cannot be shortcutted. Moreover, since the charge level remains below the charge level at v_k, shortcutting P does not create an alternative path from v_1 to v_k that improves the final charge at v_k, similarly to what happened with the path $u_1u_2u_3$ from the previous example.

We prove in Theorem F.1 that in $\tilde{O}(n^{3.5})$ time we can compute a 2-dimensional table $M[\cdot][\cdot]$ that *dominates* all simple monotone paths. That is, for every simple monotone path $P = v_1 \dots v_k$, it holds that $M[v_1][v_k] \geq g(P)$. Moreover, the table M is *sound*. That is, for every $u, v \in V$, if $M[u][v] \neq -\infty$, then there exists a monotone path P (not necessarily simple) from u to v such that $g(P) \geq M[u][v]$. Since monotone paths are traversable, it follows that if $M[u][v] \neq -\infty$, then $M[u][v] \geq -B$. Note that it is possible that $M[u][v] > B$. Once we have computed M, solving the all pairs $\alpha_B(\cdot,\cdot)$ problem is rather simple, we explain this derivation at the end of the technical review.

The following is a high level description of the computation of M. For simplicity, in this short review, we only describe how to dominate *ascending* paths. A simple observation is that every monotone path P contains a monotone subpath of edge-length 2 or 3. We call such a path a *short* monotone path. Thus, by shortcutting such a short monotone subpath into a single arc, we get an ascending path P' of smaller length than P and larger or equal gain than $g(P)$. This observation leads to a trivial $\tilde{O}(n^4)$ algorithm: Perform n iterations and generate a series of graphs $G_0 = G, G_1, \dots, G_n$. In the i'th iteration we find for every u, v the largest gain short monotone path from u to v in G_i. Once we have found all such gains, we build G_i by increasing the gains of every arc[4] (u, v) in G_{i-1} if there is a corresponding short monotone path from u to v of a better gain. We can implement each iteration in $\tilde{O}(n^3)$ time using a BST data structure. The table M stores the gains of the arcs of final graph G_n. Given a simple ascending path P in G_0, this process implicitly constructs a series of paths $P_i \in G_i$, where P_i is obtained from P_{i-1} by shortcutting as many short monotone paths as possible and P_n is a single arc.[5]

An immediate question is whether $\Theta(n)$ iterations are necessary. The answer is yes. The reason for this are *double-funnels* (see Figure 2(a)). A path $P = v_1 \dots v_k$ is a *double-funnel* if P does not contain a short monotone subpath. Double-funnels can have $\Theta(n)$ edges and an ascending monotone path which consists mainly of a long double-funnel would require $\Theta(n)$ iterations to be shortcutted into a single arc, see Figure 2(b).

As a consequence of the discussion above, in order to improve upon the simple algorithm, we need to handle double-funnels and reduce the number of iterations. A simple observation is that every ascending path can be viewed as an alternation between double-funnels (that are maximal with respect to inclusion) and short monotone paths, see Figure 3. Indeed, by the definition of a double-funnel, if we extend a double-funnel that is maximal with respect to inclusion by a single arc, the path ceases to be a double-funnel and therefore contains a short monotone path.

Let P be an ascending path such that P is not shortcutted to a single arc after $T = \sqrt{n}$ iterations of the simple algorithm. Let $P_0, \dots, P_T$ (paths in $G_0, \dots, G_T$, respectively) be the corresponding

[4]We assume G_{i-1} is a full graph by adding arcs of gain $-\infty$.

[5]Note that P_i is not uniquely defined since short monotone paths may overlap. Moreover, we might perform shortcuts in P_{i-1} because of short monotone paths that appear in G_{i-1} and not in P_{i-1}.

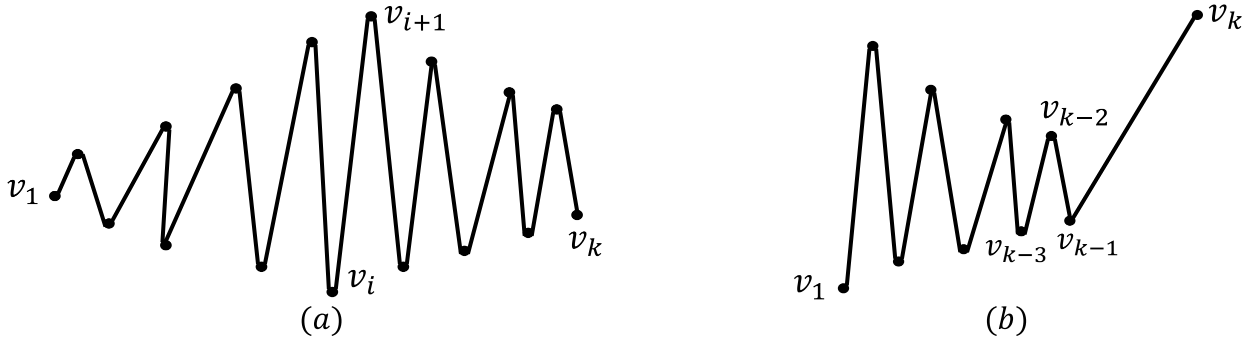


Figure 2: On the left: a double-funnel. On the right: worst case example for the simple algorithm. The depicted (directed) path $P = v_1 \dots v_k$ is monotone. Since $v_1 \dots v_{k-1}$ is a double-funnel, the only short monotone subpath of P is $v_{k-3}v_{k-2}v_{k-1}v_k$. Assume G is a path graph that contains only the path P. After the first iteration of shortcutting short monotone paths, we are left with the path $P_1 = v_1 \dots v_{v-3}v_k$ that has a similar structure to P. Thus, $\lfloor \frac{k}{2} \rfloor$ iterations are necessary in order to shortcut P into a single arc.

sequence of ascending paths as we defined before. For every $i = 0, \dots T$, denote by f_i the number of (maximal with respect to inclusion) double-funnels in P_i. By the interleaving property of double-funnels and short monotone paths, for every $i = 0, \dots, T-1$, the number of short-monotone subpaths in P_i is at least f_i and therefore $|P_{i+1}| \le |P_i| - f_i$ (where $|Q|$ denotes the number of arcs in a path Q). Since $P = P_0$ is a simple path (and thus of length at most $n-1$), and since we can uniquely charge a short monotone path that we shortcut at iteration i to each funnel in P_i it follows that $\sum_{i=1}^{T} f_i < n$, so the average number of funnels per iteration (of the T iterations that we consider) satisfies $\frac{1}{T}\sum_{i=1}^{T} f_i = \frac{1}{\sqrt{n}}\sum_{i=1}^{\sqrt{n}} f_i < \sqrt{n}$. By Markov's inequality, in at least $\frac{1}{2}T = \frac{1}{2}\sqrt{n}$ iterations, $f_i \le 2\sqrt{n}$. Thus, in at least half of the T iterations, the paths P_i have $O(\sqrt{n})$ double-funnels. By sampling uniformly at random $\Theta(\log n)$ iterations, we are guaranteed to "hit" such an iteration w.h.p.. The final component of our algorithm is the procedure *Long-Shortcuts*(G_i), that, given a path P_i with $O(\sqrt{n})$ double-funnels, finds *long* shortcuts (i.e., shortcuts that correspond to monotone paths that could be of any length) in P_i, resulting in a path P_{i+1} that is shorter than P_i by a constant factor.

Based on the above discussion, our algorithm proceeds as follows. We perform $\tilde{\Theta}(\sqrt{n})$ iterations. In each iteration we find all short monotone path and shortcut them (this results in a modified graph with larger arc gains). Moreover, in each iteration, with probability $\tilde{\Theta}(\frac{1}{\sqrt{n}})$ we additionally call *Long-Shortcuts* which finds long monotone paths in the current graph, shortcuts them, and returns a modified graph.

We now describe the procedure *Long-Shortcuts*(G_i). We extensively use two path structures in *Long-Shortcuts*: *Arc-bounded* paths and *funnels*, see Figure 1. A path $P = v_1 \dots v_k$ is first arc-bounded if for every $i = 2 \dots, k$, it holds that $\sum_{j=1}^{i-1} g(v_j v_{j+1}) \le \max\{0, g(v_1v_2)\}$ and $\sum_{j=1}^{i-1} g(v_j v_{j+1}) \ge \min\{0, g(v_1v_2)\}$. A last arc-bounded path is defined analogously. A path is arc-bounded if it is either first or last arc-bounded. A path P is a funnel if it is both arc-bounded and a double-funnel. Observe that any double-funnel can be decomposed to two funnels, each starts or ends at the edge of largest gain in absolute value, see Figure 3. Given the current graph G_i, *Long-Shortcuts*(G_i) stores a table $D[\cdot][\cdot]$ such that for every $u, v, w \in V$, $D[uv][w]$ stores the largest recorded gain of a first arc bounded path in G_i that starts with the arc uv and ends at w ($D[u][vw]$ is defined similarly for last arc-bounded paths). Algorithm *Long-Shortcuts* first generates arc-bounded paths (that is, stores values in the table D) and finally, finds long monotone paths based on those arc bounded paths. To ease the explanation, we begin by demonstrating the latter.

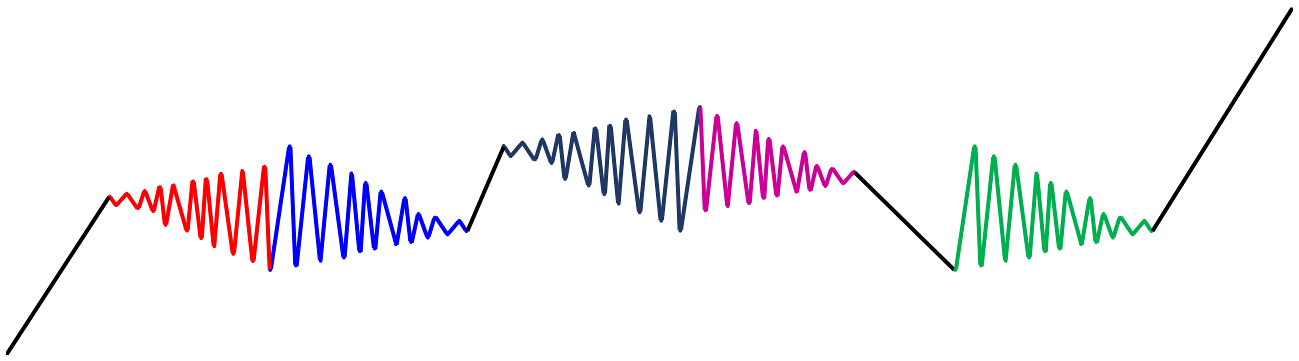


Figure 3: A decomposition of an ascending path to double-funnels that are maximal with respect to inclusion. Observe that “the gap” between two double-funnels contains a short-monotone path. The double-funnels are split into two funnels. Note that the green double-funnel is not maximal with respect to inclusion (it can be extend backwards by 2 arcs), this was done for aesthetic reasons to show “the gap” after the purple funnel.

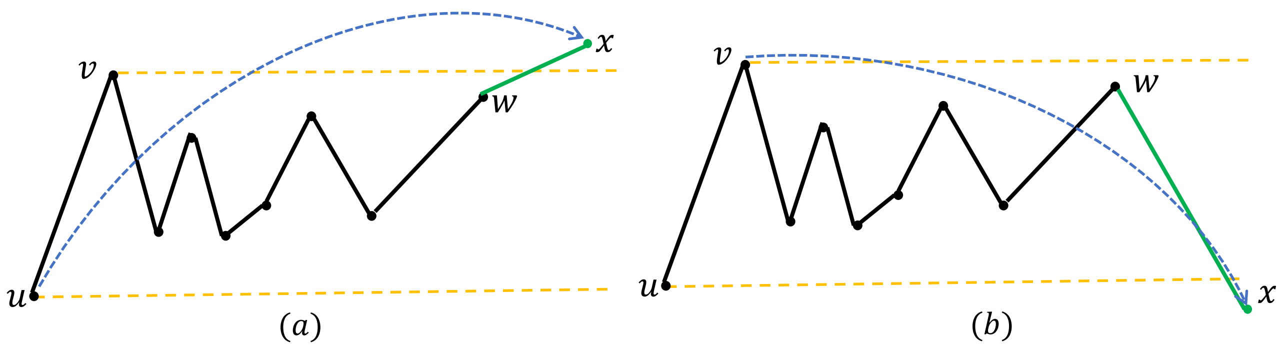


Figure 4: Finding a monotone path by extending an arc-bounded path by a single arc.

2.1 Generating monotone paths from arc-bounded paths

This part is straightforward: Given a vertex $u \in V$, we consider all arc-bounded paths that start at u and we extend each by a single arc: We scan all triplets $v, w, x \in V$, such that $D[uv][w] \neq -\infty$, and “concatenate” the arc-bounded path $P^{uv,w}$ that corresponds to $D[uv][w]$ with the arc wx, resulting in a path $P^{uv,x}$ to x that starts with uv.[6] Assume $g(uv) > 0$ (other cases are similar). If this concatenated path remains arc bounded then we did not find a monotone path. Otherwise, either $D[uv][w] + g(wx) > g(uv)$ or $D[uv][w] + g(wx) < 0$. It is easy to see that in the former case, $P^{uv,x}$ is ascending (see Figure 4(a)), and in the latter case the subpath from v to x is descending (see Figure 4(b)). It is easy to see that the running time of this process is $O(n^3)$.

2.2 Finding arc-bounded paths

As already discusses, any path can be viewed as an alternation between double-funnels (which are just two funnels that are concatenated) and short monotone paths. Thus, handling funnels has a crucial role.

We compute arc bounded paths using two building blocks.

1. A procedure *Compute-Funnels*(H) to compute funnels. Given a graph H, *Compute-Funnels*(H) returns a table $D[\cdot][\cdot]$ that dominates every funnel (which is a simple path) in H. That is, for every funnel $P = v_1 \dots v_k$ that is first arc-bounded, it holds that $D[v_1 v_2][v_k] \geq g(P)$. Similarly,

[6]We do not actually store paths. Instead we examine the quantity $D[uv][w] + g(wx)$.

for every funnel $P = v_1 \dots v_k$ that is last arc-bounded, it holds that $D[v_1][v_{k-1}v_k] \geq g(P)$. Moreover, the table D is *sound*. That is, for every $u, v, w \in V$, if $D[uv][w] \neq -\infty$, then there exists a first arc-bounded path $Q = v_1 \dots v_k$ (not necessarily a funnel) such that $g(Q) \geq D[uv][w]$. For the full details, see Appendix E.3.3

2. A concatenation procedure *Concatenate*(H, D, v). Given a graph H, a table $D[\cdot][\cdot]$ and a vertex $v \in V$. The procedure, in a brute force manner, scans all 4-tuples (w, x, y, z) of vertices and then tries to concatenate a first arc-bounded path in D that start with the arc vw and end at x with first arc-bounded path that start with the arc xy and end at z.[7] Note that this procedure only generates arc-bounded paths that start at v. For the full details, see Appendices E.3.2 and E.3.4. A naive implementation of this procedure takes $O(n^4)$ time. Using a balanced binary search tree, we get a running time of $\tilde{O}(n^3)$. Since our claimed running time for the entire algorithm is $\tilde{O}(n^{3.5})$, we can use the *Concatenate* procedure only $\tilde{O}(n^{0.5})$ times.

We now describe *Long-Shortcuts*(H) and the intuition about it. The algorithm starts by calling to *Compute-Funnels*(H), which in $\tilde{O}(n^{3\frac{1}{3}})$ time computes a table $D[\cdot][\cdot]$ that dominates all simple funnels in H. The algorithm then samples uniformly at random sets $S_i \subseteq V$ of size $\tilde{O}\left(\frac{\sqrt{n}}{2^i}\right)$, for $i = 1, \dots, \log(\sqrt{n})$. Then, for every $i = 1, \dots, \log(\sqrt{n})$ and $u \in S_i$ we perform 2^i times the procedure *Concatenate*(H, D, u). Finally, we extract monotone paths by applying the procedure from Appendix 2.1 on every vertex in $S = \cup_i S_i$.

We now give the intuition behind the algorithm. Recall the discussion about "hitting" an iteration in which $P_i = v_1 \dots v_k$ (an ascending path in G_i, for some $0 \leq i \leq T = \sqrt{n}$, that represents the evolution of $P = P_0$ over the iterations of shortcutting) has at most $2\sqrt{n}$ double-funnels. Assume we run *Long-Shortcuts*(G_i). Every vertex $v_j \in P_i$ defines a first arc-bounded path $P' = v_j \dots v_t$, where $j \leq t \leq k$ is maximal such that $v_j \dots v_t$ is first arc-bounded, see Figure 5. Note that P' may contain several double-funnels, say f. Thus, if we apply *Concatenate*(G, D, v_j), $\Theta(f)$ times, the table D will "find" P' (that is we will have $D[v_j v_{j+1}][v_t] \geq g(P')$). By the discussion in Appendix 2.1, if we extend $v_j \dots v_t$ by the arc $v_t v_{t+1}$ we will find a monotone path of length $t - j + O(1)$. For this process to be efficient, we have to balance the work we do (which is proportional to the number of funnels in P' which is the number of calls to concatenate that we need to do to find P') to compute P' with the reward we achieve (which is proportional to the length of P') by shortcutting the monotone path corresponding to P'.

We are shooting for a running time of $O(n^{3.5})$, therefore as we already said we can call concatenate at most $O(\sqrt{n})$ times (recall that it works for a single particular vertex at each call). In particular, for every $i = 1, \dots, \log(\sqrt{n})$, the product of $|S_i|$ and the number of calls of concatenate from each vertex of S_i should be $O(\sqrt{n})$. To explain why we need the $O(\log(n))$ levels of sampling, we consider the two extreme cases which our sampling interpolates between. That is, the case of $i = \log(\sqrt{n})$ where $S_i = O(1)$ and the case of $i = 1$ where $|S_i| = O(\sqrt{n})$.

These two cases are demonstrated in Figure 5 for a path P_i of length $\Theta(n)$ and $\Theta(\sqrt{n})$ funnels. The first example ($i = \log(\sqrt{n})$), depicted in Figure 5(a), considers the case in which all funnels, except for the first one, are of constant length and the rest is filled with the first funnel which is of linear size. Moreover, the arc-bounded paths that correspond (in the manner explained in the previous paragraph) to every vertex in a short funnel are of constant length and the arc-bounded paths that correspond to vertices in the long funnel are all reaching the last arc of the path. Thus, in order to achieve sufficient reward (i.e., find long enough monotone paths), we have to sample a vertex u in the long funnel and then perform $\Theta(\sqrt{n})$ times *Concatenate*(G_i, D, u). Thus, the example shows that there are cases in which we have to perform $\Theta(\sqrt{n})$ concatenations at a single vertex.

[7]Formally, we look on the quantity $D[vw][x] + D[xy][z]$ and verify some inequalities to make sure that the concatenated path is indeed first arc-bounded.

The second extreme case, depicted in Figure 5(b), is the case in which all funnels are of length $\Theta(\sqrt{n})$ and for every $v \in P_i$, the arc-bounded path that corresponds to v contains a single funnel. Thus, for every $v \in P_i$ we can apply a single concatenation and find the arc-bounded path that corresponds to v and later extend it to a monotone path of length $O(\sqrt{n})$. In this case to reduce the length of P_i by a constant factor, we have to sample $\Theta(\sqrt{n})$ vertices (that will hit a constant fraction of the funnels) and perform a constant number of concatenation on each one of them.

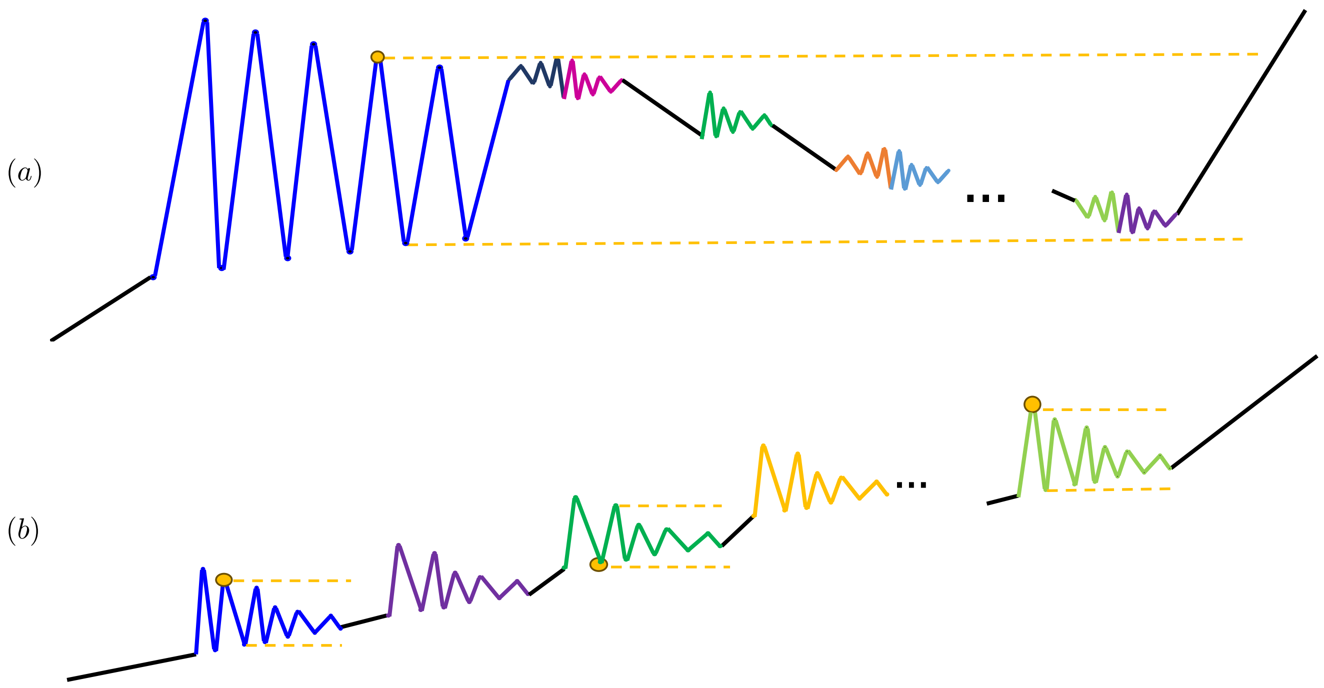


Figure 5: Two extreme cases for algorithm *Long-Shortcuts*. Black lines represent single arcs. Figure (a) shows why we need to sample $O(1)$ vertices but perform $\Theta(\sqrt{n})$ concatenations per vertex. Figure (b) shows why we need to sample $\Theta(\sqrt{n})$ vertices but perform $O(1)$ concatenations per vertex.

2.3 Solving the all-pairs problem

Finally, we briefly describe the key observations that relate monotone paths to the computation of $\alpha_B(\cdot,\cdot)$. We begin by assuming that the optimal energetic paths are simple and later show how to solve the general case in which the optimal paths use positive cycles.

2.4 Simple energetic paths

Assume we have computed the table $M[\cdot][\cdot]$ that dominates every simple monotone path in G. Let $s, t \in V$ and let $P = v_1 \dots v_k$ be an optimal energetic path from $v_1 = s$ to $v_k = t$ (that is, $\alpha_B(s,t) = \alpha_B(P)$). We consider the special case in which P is simple and for every $1 < i \leq k$ it holds that $\alpha_B(v_1 \dots v_i) < B$. That is, the car starts with full charge at s and its charge level remains below B. We decompose P as follows. Let $v_{i_1} = s$ and let v_{i_2} be the vertex of lowest gain in P. We define v_{i_3} to be the vertex of highest gain in the suffix $v_{i_2} \dots v_k$ and so on, see Figure 6(a). This results in a series of vertices $s = v_{i_1}, v_{i_2}, \dots, v_{i_r} = t$. Clearly, this partitioning divides P into monotone segments that alternate between ascending and descending paths. A key observation is that these monotone paths are optimal in terms of gain. That is, for every $1 \leq j < r$, there is no monotone path Q from v_{i_j} to $v_{i_{j+1}}$ with larger gain than the subpath $v_{i_j} \dots v_{i_{j+1}}$. Otherwise, we can replace the subpath

$v_{i_j} \dots v_{i_{j+1}}$ by Q and increase the final charge at t,[8] a contradiction to the optimality of P. Thus, for every $1 \le j \le r$, it holds that $M[v_{i_j}][v_{i_{j+1}}] = g(v_{i_j} \dots v_{i_{j+1}})$. Let G' be a directed clique whose gains are defined by $M[\cdot][\cdot]$. The final observation is that $v_{i_1} v_{i_2} \dots v_{i_r}$ is a funnel in G'. Thus, by calling *Compute-Funnels*(G') we can find this funnel.

2.5 Handling positive cycles

A simple observation is that every positive gain cycle C contains a pair of points $x, y \in C$ such that the car can start at x with zero charge, and traverse the cycle until it reaches y with a fully charged battery (i.e., B charge).[9] We say that (x, y) is an *entry-exit* pair of C, where x is the *entry* and y is the exit.

We prove in Lemma H.6, that every positive cycle C contains an entry-exit pair (x, y) such that C^{xy}, the path from x to y through C, is ascending and C^{yx}, the path from y to x through C, is descending.[10] This lemma, leads to a simple algorithm for identifying entry-exit pairs: For every $x, y \in V$, if $M[x][y] > 0$ and $M[x][y] + M[y][x] > 0$, then set $\alpha_0(x, y) = B$ (i.e., (x, y) is an entry-exit pair). The positive shortcut $M[x][y]$ indicates that there is an ascending path P^{xy} from x to y. If $M[x][y] \ge B$ then clearly we can start at x with zero charge and get to y with full charge (by using the shortcut[11] xy of gain $M[x][y]$). Otherwise, the second inequality $M[x][y] + M[y][x] > 0$ guarantees that we can start at x with zero charge and get back to x with positive charge (by using the shortcuts xy and yx). Therefore, by extending the path to y, we generate an ascending path with larger gain $M[x][y] + M[y][x] + M[x][y] > M[x][y]$, see Figure 6$(b)$. By repeating this multiple times, we get an ascending path from x to y with gain larger than B justifying setting $\alpha_0(x, y) = B$.

We perform 3 additional simple inferences: For every $x, y, z \in V$

- If $M[x][y] + M[y][z] \ge 0$ and $M[x][y] \ge 0$, we deduce that the path that consists of the two shortcuts $M[x][y], M[y][z]$ is a witness that $\alpha_0(x, z) \ge 0$. That is, it is possible to start at x with zero charge and reach z: Either $M[x][y] \ge B$ and then the claim follows by the traversability of monotone paths ($M[y][z] \ge -B$) or $M[x][y] < B$ and therefore either $M[y][z] \ge 0$ or $-M[x][y] \le M[y][z] < 0$. The former case is trivial. In the latter case, we can start with zero charge at x and reach y with $M[x][y]$ charge and then continue to z and reach it with $M[x][y] + M[y][z] \ge 0$ charge.
- If $M[x][y] + M[y][z] \ge 0$ and $M[y][z] \ge 0$, we deduce that $\alpha_B(x, z) = B$.
- If $M[x][y] \ne -\infty$ (so $M[x][y] \ge -B$), we infer that $\alpha_B(x, y) \ge 0$. That is, it is possible to reach y if we start at x with full charge.

Finally, we combine these relations into a graph H and compute its transitive closure $H^\star$. The graph H is defined as follows. $H = (V^0 \cup V^B, E(H))$, where $V^0 = \{v^0 \mid v \in V\}$ and $V^B = \{v^B \mid v \in V\}$ are two copies of V. Each vertex $v^0 \in V^0$ represents being at v with 0 charge and each vertex $v^B \in V^B$ represents being at v with full charge. An arc $u^{b_1} v^{b_2} \in E(H)$ represents that $\alpha_{b_1}(u, v) \ge b_2$.[12] We create the arcs $E(H) \subseteq \{u^{b_1} v^{b_2} \mid \alpha_{b_1}(u, v) \ge b_2\}$ according to the 4 relations shown above (for example, if $M[x][y] \ne -\infty$, we add the arc $x^B y^0$ to H). We claim in Theorem H.12 that, for every $s, t \in V$, $\alpha_B(s, t) = B$ if and only if $s^B t^B \in E(H^\star)$.

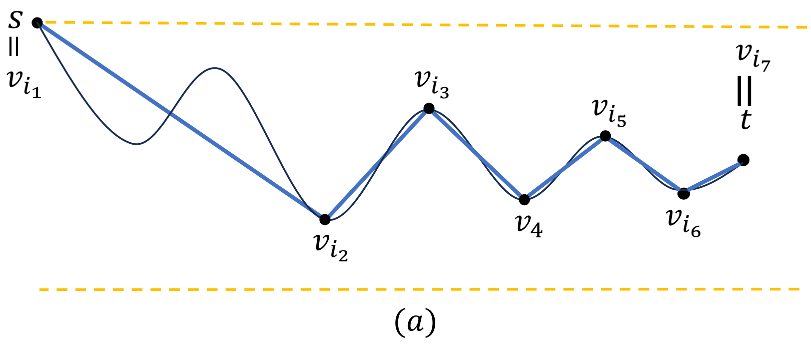
[8] We use here the fact that the battery is not full.

[9] It is possible that the car took the direct path in C from x to y, or it cycled through C several times.

[10] It is possible that $x = y$. For example in a cycle in which all arc gains are positive.

[11] Recall that using shortcuts does not change the α values since each shortcut corresponds to a monotone path in G of the same gain.

[12] Note that the other direction does not necessarily hold: It is possible that $\alpha_{b_1}(u, v) \ge b_2$ but $u^{b_1} v^{b_2} \notin E(H)$.



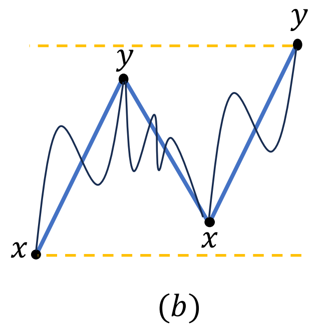


Figure 6: (a) A decomposition of an optimal path from s to t into a sequence of simple monotone paths. After shortcutting these paths, we are left with a funnel. (b) Illustration of why $M[x][y] + M[y][x] > 0$ & $M[x][y] > 0$ leads to $\alpha_0(x, y) = B$. Each blue arc represents a shortcut in M. Each such shortcut can be unwrapped into a path in G

Using the graph $H^\star$, our algorithm reduces the all pairs $\alpha_B(\cdot,\cdot)$ problem to the case in which the energetic paths are simple: For every $s, t \in V$, using $H^\star$, we find all vertices $x \in V$ such that $\alpha_B(s, x) = B$ and then, as in Appendix 2.4, we find the best energetic simple path from any such x to t.

The following is a brief review of the correctness of the algorithm. Let $s, t \in V$ and let $P = v_1 \dots v_k$ be an optimal energetic path from s to t (i.e., $\alpha_B(s, t) = \alpha_B(P)$). We argue that there is a vertex x on P such that $\alpha_B(s, x) = B$ and $\alpha_B(x, t) = \alpha_B(s, t)$. If $\alpha_B(s, t) = B$, then we are done since this relation is already recorded in $H^\star$ and we can set $x = t$. Otherwise, let $1 \le i \le k$ be maximal such that $\alpha_B(v_1 \dots v_i) = B$. It follows that $\alpha_B(s, v_i) = B$ and for every $i < j \le k$ it holds that $\alpha_B(v_1 \dots v_i) < B$. This implies that $v_i \dots v_k$ must be a simple path.[13] So we conclude that the algorithm finds the optimal energetic path when inspecting $x = v_i$.

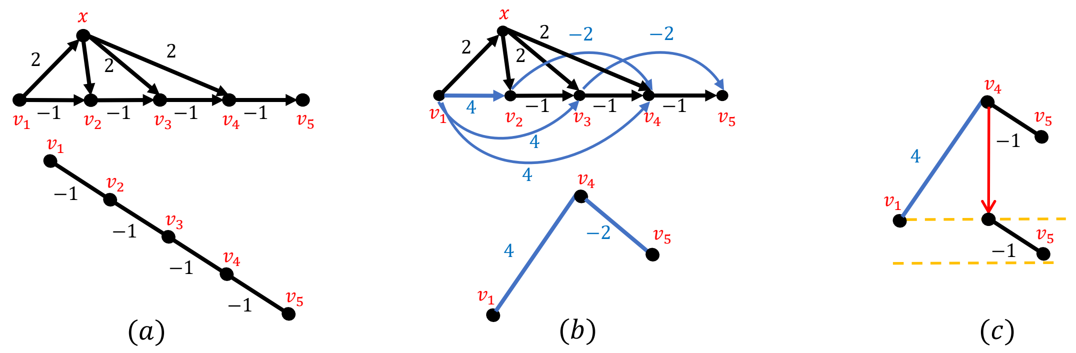
2.6 A technicality - charge drop schedules

In this section we describe *Charge drop schedules* and the technical challenge that it addresses. Before we delve into the definition, we motivate it by pinpointing several problems with our arguments.

1. Throughout this section we explained how to shortcut an ascending path to single arc via a sequence of short/long shortcut updates. A key invariant that is required for this argument to hold is the fact that given an ascending path $P = v_1 \dots v_k$, if we replace a monotone subpath $v_i \dots v_j$ of P by a monotone path Q of larger gain, then the resulting path $P' = v_1 \dots v_i \mid Q \mid v_j \dots v_k$ (The $\mid$ stands for concatenation) is ascending and $g(P') > g(P)$. Unfortunately, this argument does not hold if P is descending. For example, consider the graph G in Figure 7(a) and the descending path $P = v_1v_2v_3v_4v_5$. After performing one iteration of the simple algorithm (computing all short monotone paths and updating the gains of the graph), we are left with a graph G' with gain function g' (see Figure 7(b)) that does not contain any monotone path from v_1 to v_5. This is of course unsettling, as finding the best short shortcuts should be a good property of the algorithm and yet it destroyed some other descending paths

2. Recall the procedure *Concatenate*(G, D, v) that scans all 4-tuples (w, x, y, z) of vertices and then tries to concatenate a first arc-bounded path (stored in D) that starts with the arc vw and ends at x with first arc-bounded path that starts with the arc xy and ends at z (which is done by calculating $D[vw][x] + D[xy][z]$ and verifying some inequalities). Consider the following example: Assume $g(vw) = 5, g(xy) = 3$ and $D[vw][x] = 2, D[xy][z] = 3$. Therefore,

[13] Otherwise, $v_i \dots v_k$ contains a positive cycle, so by repeating the cycle (and using the fact that no vertex on cycle, and the rest of the path, has already reached full charge) we can increase the final charge at $v_k = t$, a contradiction.



x
2
2
2
v_1 -1 v_2 -1 v_3 -1 v_4 -1 v_5
v_1 -1 v_2 -1 v_3 -1 v_4 -1 v_5
(a)
x
2 2 2 -2 -2
v_1 4 v_2 -1 v_3 -1 v_4 -1 v_5
4
4
v_4
4
-2
v_5
v_1
(b)
v_4
v_5
4
-1
v_1
-1
v_5
(c)

Figure 7: (a) The graph G and the descending path $P = v_1v_2v_3v_4v_5$. (b) The graph G' that we get after shortcutting all short monotone paths. Blue arcs correspond to either new arcs or arcs with increased gain. Note that there is no monotone path from v_1 to v_5 in G'. (c) By using charge drop schedule, we can transform the path $v_1v_3v_5$ into a short descending path of gain -2.

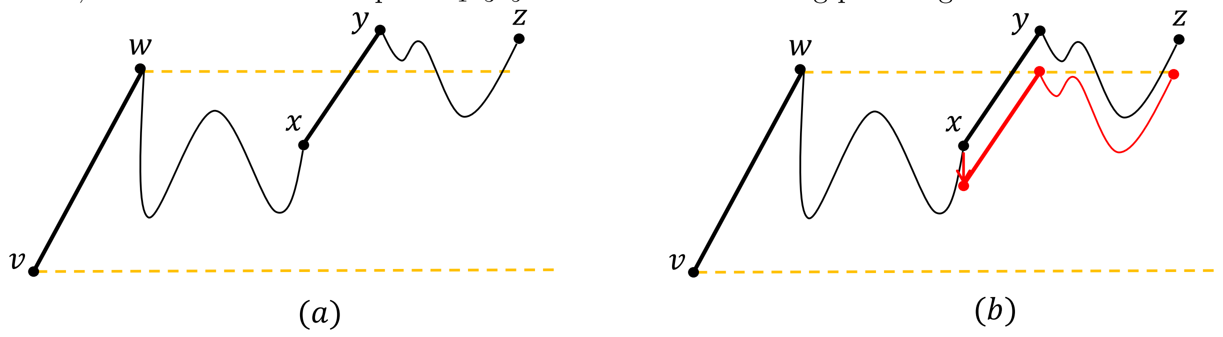


Figure 8: A use case of charge drops. (a) Two arc-bounded paths whose concatenation is not arc-bounded. (b) By applying a simple charge-drop schedule we make the concatenated path arc-bounded.

by running *Concatenate*(G, D, v), we will concatenate the arc-bounded paths corresponding to $D[vw][x]$ and $D[xy][z]$ and get an arc bounded path that starts at vw and ends at z with gain $D[vw][z] = D[vw][x] + D[xy][z] = 5$. Unfortunately, this concatenation is not guaranteed to happen. It is possible that earlier in the run of *Concatenate*(G, D, v), the algorithm managed to improve $D[vw][x]$ to $D[vw][x] = 3$ and therefore concatenating $D[vw][x]$ to the arc-bounded path corresponding to $D[xy][z]$ does not result anymore in an arc-bounded path, see Figure 8(a). Again, by performing an update that should be good for us (increasing $D[vw][x]$ from 2 to 3), we hurt ourself somewhere else (we did not make the update $D[vw][z] = 5$).

In both examples, we suffered from having computed values that are "too good". The simple concept that solves this problem is *charge drop*. Charge drops allow us, at any vertex along the path, to get rid of some charge, see Figure 9. Formally, let $P = v_1 \dots v_k$ be a path in G. A charge drop schedule is a vector $C = (d_1, d_2, \dots, d_k) \in \mathbb{R}^k_{\geq 0}$, where $d_1 = 0$. The *gain* at v_i with respect to P and C, denoted as $g^{P,C}_{v_i}$ is defined as $g^{P,C}_{v_i} = \sum_{t=1}^{i-1} g(v_tv_{t+1}) - \sum_{t=2}^{i} d_t$, for $2 \leq i \leq k$ and $g_{v_1} = 0$ otherwise. Monotone paths and arc bounded paths can be defined similarly to before by replacing the gain of an arc $g(v_iv_{i+1})$ by $g(v_iv_{i+1}) - d_{i+1}$. When P is clear from contexts, we abbreviate $g^{P,0}_{v_i}$ and write g_{v_i}. We now show how to fix the two examples using charge drop schedules.

1. In the first example (see Figure 7) $P = v_1v_2v_3v_4v_5$ is a descending path in G, but there is no descending (or ascending) path from v_1 to v_5 in G'. Instead, G' contains the path $v_1v_4v_5$ that has positive gain. By using a simple charge drop schedule that drops 4 units of charge at v_4, we view $v_1v_4v_5$ as a short descending path of gain -2, see Figure 7(c).

2. In the second example we faced a problem when trying to concatenate an arc-bounded path corresponding to $g(vw) = 5, D[vw][x] = 3$ and an arc bounded path corresponding to $g(xy) = 3, D[xy][z] = 3$. By simply dropping a single unit of charge at x (the concatenation point), we are now able to concatenate the two paths and therefore assign $D[vw][z] = (D[vw][x]-1)+D[xy][z] = 5$, see Figure 8.

We incorporate charge drops in our algorithm in the following places.

1. When computing all short monotone paths, if a path P (of length 2 or 3) starts by a negative gain arc, we will always apply charge drop schedule and create a descending path out of P. For example, if $P = v_1v_2v_3v_4$ and $g(v_1v_2) = -5, g(v_2v_3) = 2, g(v_3v_4) = -1$, then we record a descending path from v_1 to v_4 of gain -5 (this corresponds to dropping one unit of charge at v_4).

2. In the computation of long monotone paths. Recall that we consider tuples $u, v, w, x \in V$ and we extend the arc-bounded path that corresponds to $D[uv][w]$ by the arc wx. We incorporate charge drops in the following case: If $g(uv) < 0$ and $D[uv][w] + g(wx) \in [g(uv), 0]$ (that is the concatenated path remains arc-bounded), we record a descending path from u to x of gain $g(uv)$. This corresponds to performing a charge drop at x that drops $D[uv][w] + g(wx) - g(uv)$ charge.

3. In the concatenation procedure, whenever the concatenation of the two arc bounded paths does not yield an arc-bounded path, we perform a charge drop to force the result to be arc-bounded. That is, for every $v, w, x, y, z \in V$, if $g(vw) > g(xy) > 0$ and $D[vw][x] + D[xy][z] > g(vw)$, we set $D[vw][z] = g(vw)$. This corresponds to performing the smallest possible charge drop at x such that the concatenated path is arc-bounded, see Figure 8(b).

2.7 Main technical lemma

In this section, we prove a simplified version[14] of our main lemma (Lemma F.2). Recall our algorithm: We perform $\tilde{\Theta}(\sqrt{n})$ iterations. In each iteration we find all short monotone path and shortcut them (this results in a modified graph with larger arc gains). Moreover, in each iteration, with probability $\tilde{\Theta}(\frac{1}{\sqrt{n}})$ we additionally call *Long-Shortcuts* which finds long monotone paths in the current graph, shortcuts them, and returns a modified graph.

Lemma 2.1. *Let $P = v_1 \dots v_k$ be a simple ascending path in G. Let G' be the modified graph after $\sqrt{n}$ iterations of the modified algorithm and let g' be its gain function. If $|P| \le \sqrt{n}$, then $g'(v_1v_k) \ge g(P)$. If $|P| > \sqrt{n}$, then w.h.p. there is an ascending path P' in G' from v_1 to v_k in that satisfies $g'(P') \ge g(P)$ and $|P'| \le (1 - 1/\Omega(\log n)) \cdot |P|$.*

Lemma 2.1 is derived from Lemma 2.2, which is our main technical lemma. It provides guarantees about *Long-Shortcuts*, when run on a graph with an ascending path that contains few double-funnels.

Lemma 2.2. *Let $P = e_1 \dots e_k$ be a simple ascending path in G from x to y. Let $t(\ge 1)$ be the number of double-funnels in P that are maximal with respect to inclusion. Let G' be the updated graph resulted from Long-Shortcuts(G).*[15] *If $t \le k/\sqrt{n}$, then w.h.p. there is an ascending path P' in G' from x to y that satisfies $g^{G'}(P') \ge g^G(P)$ and $|P'| \le (1 - 1/\Omega(\log n)) \cdot |P|$.*

We prove Lemma 2.2 at the end of this section. The derivation of Lemma 2.1 is now straightforward.

Proof. TOPROVE 0 □

[14] We address only ascending paths.

[15] Note that every non empty path contains at least one double-funnel.

Before proving Lemma 2.2, we need to introduce the following structural definitions. These definitions allow us to measure how many applications of *Concatenate* are needed in order to dominate an arc bounded path.

Definition 2.3. *Let $P = e_1 \ldots e_k$ be a path in G. For every $1 \le i \le k$ we define $s^P(i) \ge i$ to be the maximal index such that $e_i \ldots e_{s^P(i)}$ is first arc-bounded. When P is clear from the context, we abbreviate and write $s(i)$.*

Definition 2.4. *Let $P = e_1 \ldots e_k$ be a path in G. For every i, we define $f^P(i)$ as the number of first arc-bounded funnels in $e_i \ldots e_{s(i)}$ that are maximal with respect to inclusion. When P is clear from context, we abbreviate and write $f(i)$.*

The following lemma proves that for every path $P = e_1 \ldots e_k$, the set of paths $\{e_i \ldots e_{s(i)} \mid 1 \le i \le k\}$ is laminar. We defer the proof of this lemma to the appendix (see Lemma F.10).

Lemma 2.5. *Let $P = e_1 \ldots e_k$ be a path in G, then the set of intervals $\{(i, s(i)) \mid 1 \le i \le k\}$ is laminar.*

We are now ready to prove Lemma 2.2.

Proof. TOPROVE 1 □

3 Concluding remarks

We presented a randomized $\tilde{O}(n^{3.5})$-time algorithm for the finding optimal energetic paths between all-pairs of vertices in a weighted directed n-vertex graph with positive and negative gains that may contain positive-gain cycles. This improves upon a previous $\tilde{O}(mn^2)$-time algorithm by Dorfman et al. [5]. The new algorithm is quite involved and requires the introduction of many new ideas. Improving the running time of the algorithm is a natural open problem.

References

[1] Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The shortest path problem revisited: Optimal routing for electric vehicles. *KI*, 6359:309–316, 2010.

[2] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[3] Lubos Brim and Jakub Chaloupka. Using strategy improvement to stay alive. *Int. J. Found. Comput. Sci.*, 23(3):585–608, 2012.

[4] Dani Dorfman, Haim Kaplan, Robert E. Tarjan, Mikkel Thorup, and Uri Zwick. Minimum-cost paths for electric cars. In *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 374–382. SIAM, 2024.

[5] Dani Dorfman, Haim Kaplan, Robert Endre Tarjan, and Uri Zwick. Optimal energetic paths for electric cars. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, pages 42:1–42:17, 2023.

[6] Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011.* AAAI Press, 2011.

[7] Lester R. Ford. Network flow theory. Technical Report Paper P-923, RAND Corporation, Santa Monica, California, 1956.

[8] Loïc Hélouët, Nicolas Markey, and Ritam Raha. Reachability games with relaxed energy constraints. *arXiv preprint arXiv:1909.07653*, 2019.

[9] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[10] Samir Khuller, Azarakhsh Malekian, and Julián Mestre. To fill or not to fill: The gas station problem. *ACM Transactions on Algorithms (TALG)*, 7(3):1–16, 2011.

[11] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 664–673, 2014.

A Full Version

This appendix contains the full technical details of the paper and is organized as follows. In Appendix B we begin with some preliminary material. Appendix D then gives an overview of the algorithm. The new algorithm is composed of two stages. In Stage I, described in Appendix E, sufficiently many shortcuts are found. The correctness of Stage I is proved in Appendix F. Stage II, described in Appendices G and H, uses the shortcuts found in stage I to find the $\alpha_B(s,t)$ values, and an implicit representation of the optimal energetic paths.

B Preliminaries

Let $G=(V,A,g)$, where $g: A \to \mathbb{R}$ is a gain function. Fix the battery capacity $B>0$. Suppose we traverse a path $P=v_1u_2\ldots v_k$ starting with a charge of b at v_1. We define $\alpha_b(P)\le B$ to be the amount of charge with which we reach v_k. If P cannot be traversed with this initial charge, we let $\alpha_b(P)=-\infty$. For $s,t\in V$ and $b\in[0,B]$, define $\alpha_b(s,t)=\max\{\alpha_b(P)\mid P \text{ is a path from } s \text{ to } t\}$, i.e., the maximal final charge possible at t when starting at s with b charge. It is proved in [5] that the max in this definition is well-defined. (Note that the maximum is over a possibly infinite collections of paths, since the paths are not necessarily simple.) A path $P=v_1\ldots v_k$ is optimal if $\alpha_B(v_1,v_k)=\alpha_B(P)$. The all-pairs maximum final charge problem is to compute $\alpha_B(s,t)$ for every pair $s,t\in V$. We say that a path P is *traversable* if $\alpha_B(P)\ge 0$, that is there is some energy level that we can start with and traverse P. We say that a path P is strongly traversable if $\alpha_0(P)\ge 0$. We let $|P|$ be the *length* of P, i.e., the number of arcs in P.

The gain of an arc $uv\in A$ is $g(uv)$. The gain of a vertex v in a path P is the sum of gains of the arcs that lead to v in P. During our analysis we allow ourselves to dispose of some charge while traversing a path. This leads to the following definition of gains on paths that takes into account charge drops, see Figure 9.

Definition B.1 (Gain)**.** *Let $G=(V,A,c)$. Let $P=v_1\ldots v_k$ be a path in G and let $C=(0,d_2,\ldots,d_k)\in\mathbb{R}^k_{\ge 0}$ be a charge drop schedule.*[16] *The gain at v_i with respect to P and C, denoted as $g^{P,C}_{v_i}$ is defined as $g^{P,C}_{v_i}=\sum_{t=1}^{i-1} g(v_tv_{t+1})-\sum_{t=2}^{i} d_t$, for $2\le i\le k$ and $g_{v_1}=0$ otherwise. That is, d_t is the charge drop performed at v_t for $2\le t\le k$. We omit P and C and write g_{v_i} when P,C are clear from the context. The gain of P with respect to C, denoted $g^C(P)$, is defined to be $g^{P,C}_{v_k}$. When no charge drop schedule is introduced, then we assume that the schedule is zero: $C=(0,\ldots,0)\in\mathbb{R}^k_{\ge 0}$.*

Note that unlike the definition of the charge level of the electric car, the above definition allows the gains of vertices on a path P to be larger than B and smaller than $-B$.[17] Our algorithm, however, does not compute paths (and even subpaths) of gain smaller than $-B$.

[16] Note that $d_1=0$, i.e., we do not drop charge at the first vertex.

[17] Note that a path of gain smaller than $-B$ is not traversable.

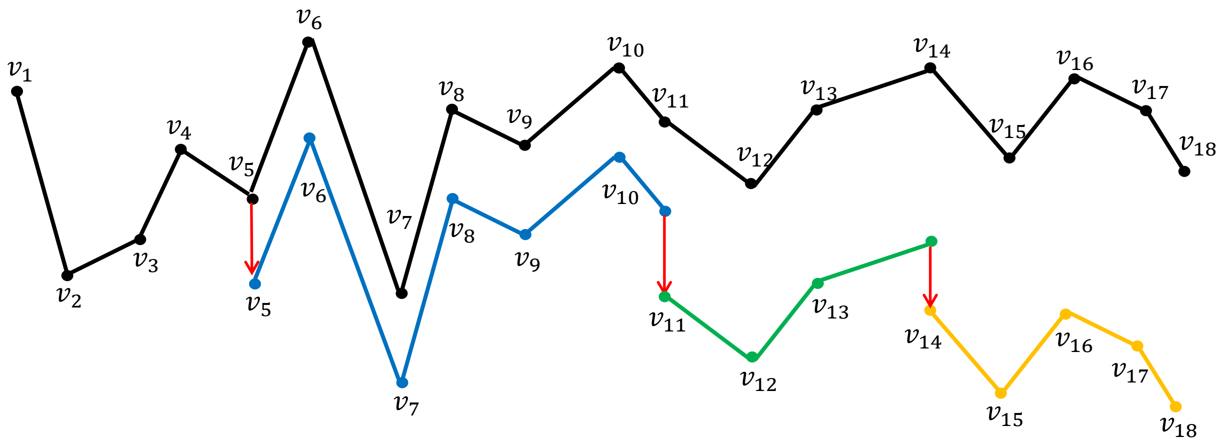


Figure 9: In black: The original gains of $P = v_1 \dots v_{18}$. Red downward arrows correspond to charge drops. In color: the gains of P with respect to the charge drops. After each charge drop we switch color. Note that each colored path has matching gains to those of a corresponding subpath of P.

Throughout this paper charge drops are used by the algorithm only twice, in Appendices E.3.5 and E.3.6. It may be instructive for a reader to first think of the case where all charge drops are 0. In the following sections we define path structures that are studied throughout the paper.

B.1 Monotone Paths and Shortcuts

Definition B.2 (Monotone path)**.** *Let $P = v_1 \dots v_k$ be a traversable path in G and let C be a charge drop schedule for P.*

- *We say that P is ascending with respect to C if $0 = g^C_{v_1} \le g^C_{v_i} \le g^C_{v_k}$, for every $1 \le i \le k$. See Figure 1(a).*
- *We say that P is descending with respect to C if $0 = g^C_{v_1} \ge g^C_{v_i} \ge g^C_{v_k}$, for every $1 \le i \le k$. See Figure 1(b).*

We say that P is monotone with respect to C if it is either ascending or descending with respect to C. We say that P is monotone if it is monotone with respect to the zero schedule.

Note that all ascending paths are strongly traversable. Also note that an ascending path might have a descending subpath and vice versa.

Lemma B.3. *If a path $P = v_1 \dots v_k$ is ascending with respect to a charge drop schedule C, then P is ascending with respect to the zero schedule.*

Proof. TOPROVE 2 □

Definition B.4 (Shortcut)**.** *We define an arc $e = xy$ (not necessarily in A) to be a k-*shortcut *in G if there is a path $P = v_1 \dots v_k$ from x to y in G which is monotone with respect to a charge drop schedule C. We say that the gain of the shortcut is $g(e) = g^C(P)$. We say that e is a shortcut in G if it is a k-shortcut in G for some k. The shortcut e is ascending if P is ascending and descending if P is descending. We say that e is a short shortcut if it is a k-shortcuts for $k \in \{2, 3\}$.*

Note that we may have parallel shortcuts corresponding to different paths, but in this case we only keep the one of largest gain.

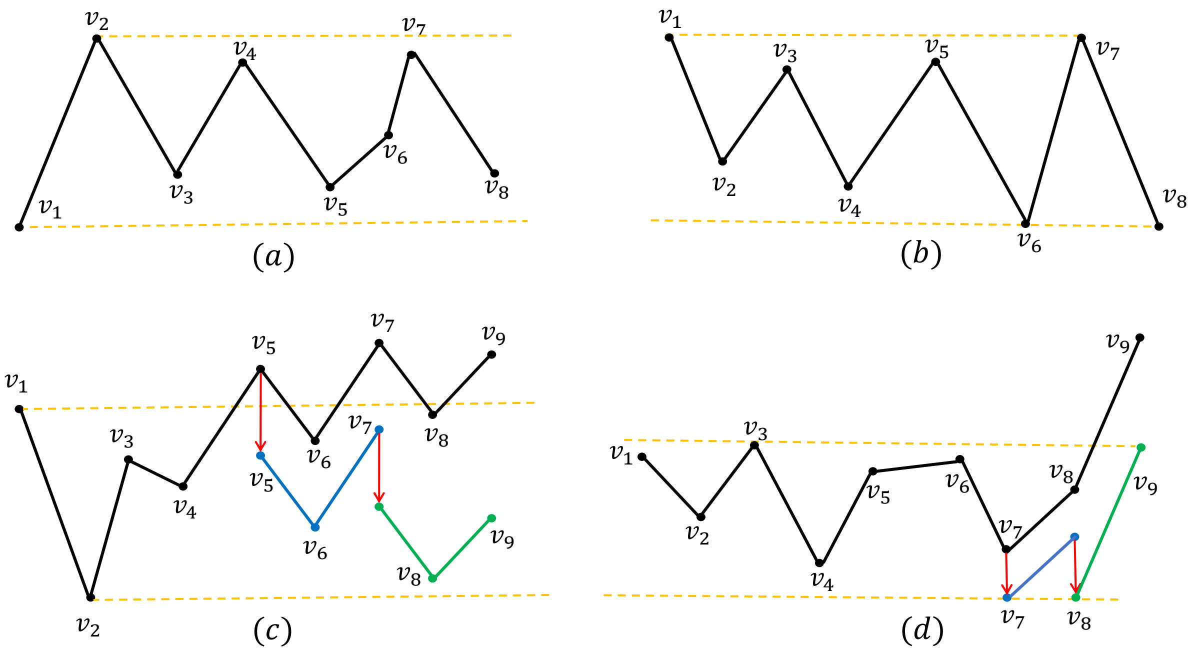


Figure 10: On the top: First-arc (left) and last-arc (right) bounded paths. Both paths are arc bounded paths with respect to the zero schedule. On the bottom: First-arc (left) and last-arc (right) bounded paths with respect to different charge drop schedules.

It is convenient to think of A as a clique where some arcs may have gain $-\infty$. Our algorithms are going to compute sets of shortcuts in some base graph G. Based on such a set of shortcuts S, it constructs a new graph G' in which $g(xy)$ for every arc xy is the maximum between $g(xy)$ in G and the gain of the shortcut xy in S. Our definitions of gain apply to the original graph or any graph that we obtain when using this procedure.

The following lemma states a core concept of our shortcutting algorithm: Every monotone path has a subpath that is a short monotone path.

Lemma B.5. *Every monotone path $P = v_1 \dots v_k$ with respect to a charge drop schedule C, where $t > 1$, contains a short shortcut with respect to C.*

Proof. TOPROVE 3 □

The following lemma shows the relation between paths that reach full charge when stating with zero charge, to ascending paths.

Lemma B.6. *Let P be a path from x to y. If $\alpha_0(P) = B$, i.e., P is strongly traversable and it reaches y with full charge, then P is ascending.*

Proof. TOPROVE 4 □

B.2 Arc-Bounded Paths

We next define arc-bounded paths, a core structure of our algorithm. A path $P = v_1 \dots v_k$ is first-arc bounded if the gain of every $v \in P$ is between the gains of the first two vertices, see Figure 10(a)-(b). We also defined arc-bounded paths with respect to charge drop schedules, see Figure 10(c)-(d).

Definition B.7 (Arc-bounded path)**.** *A path $P = v_1 \dots v_k$ is first-arc-bounded, or alternatively v_1v_2-bounded with respect to a charge drop schedule C if C does not drop charge at v_2*[18] *and if one of the following holds*

- *$g(v_1v_2) \geq 0$ and $0 = g^{P,C}_{v_1} \leq g^{P,C}_{v_i} \leq g^{P,C}_{v_2} = g(v_1v_2)$, for every $1 \leq i \leq k$. We say that P is a $\underline{v_1}\overline{v_2}v_k$ path with respect to C.*
- *$g(v_1v_2) \leq 0$ and $g(v_1v_2) = g^{P,C}_{v_2} \leq g^{P,C}_{v_i} \leq g^{P,C}_{v_1} = 0$, for every $1 \leq i \leq k$. We say that P is a $\overline{v_1}\underline{v_2}v_k$ path with respect to C.*

Similarly, P is last arc-bounded, or alternatively $v_{k-1}v_k$-bounded with respect to C if C does not drop charge at v_1 and v_{k-1} and v_k and if one of the following holds

- *$g^C(v_{k-1}v_k) \geq 0$ and $g^{P,C}_{v_{k-1}} \leq g^{P,C}_{v_i} \leq g^{P,C}_{v_k}$, for every $1 \leq i \leq k$. We say that P is a $v_1\underline{v_{k-1}}\overline{v_k}$ path with respect to C.*
- *$g^C(v_{k-1}v_k) < 0$ and $g^{P,C}_{v_k} \leq g^{P,C}_{v_i} \leq g^{P,C}_{v_{k-1}}$, for every $1 \leq i \leq k$. We say that P is a $v_1\overline{v_{k-1}}\underline{v_k}$ path with respect to C.*

We say that P is arc-bounded if it is either first-arc-bounded or last-arc-bounded. We say that P is negative arc-bounded if the "bounding" arc is of negative gain.

B.3 Funnels

The following definition defines the structure *funnel*, see Figure 1(c)-(f). Funnels are defined with respect to the zero charge drop schedule.

Definition B.8 (Funnels)**.** *A path P is said to be a* funnel *if it is arc-bounded with respect to the zero schedule and does not contain any monotone path of length 2 or 3.*

Lemma B.9. *Let $P = v_0 \dots v_k$ and denote $e_i = v_{i-1}v_i$ for $i = 1, \dots k$. P is a funnel if and only if the following two conditions hold.*

1. - *If P is e_1-bounded then $|g(e_1)| \geq |g(e_2)| > \dots |g(e_k)| > 0$, or*
   - *If P is e_k-bounded then $|g(e_k)| \geq |g(e_{k-1})| > \dots > |g(e_1)| > 0$.*

   Note that all inequalities are strict except the first.
2. *The sign of the arc gains are alternating, i.e., $sign(g(e_i)) = (-1)^{i+1} \cdot sign(g(e_1))$ for every $1 \leq i \leq k$.*

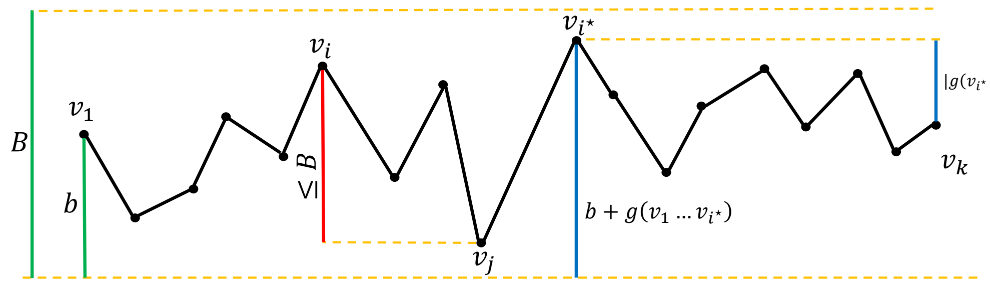
Proof. TOPROVE 5 □

As an immediate corollary of the above structural lemma, we observe that a subpath of a funnel is also a funnel.

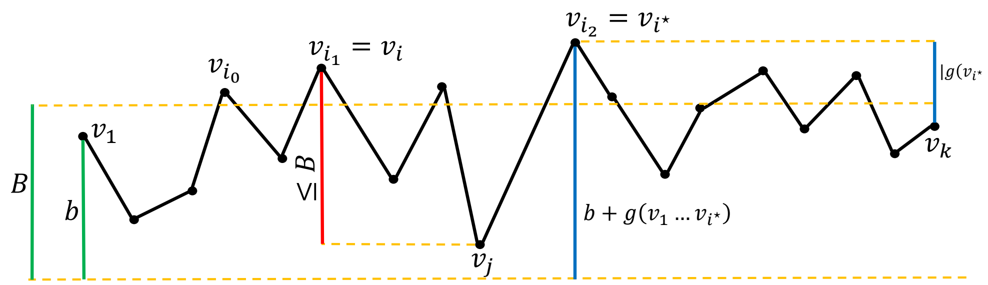
C Relating the Path Structures to $\alpha(\cdot,\cdot)$

We present several lemmas that relate monotone paths, arc-bounded paths and funnels to the $\alpha(\cdot,\cdot)$ values of G. We start with the following lemma that characterizes traversable paths. It states that a path is traversable if and only if it has no subpath that loses more than B gain (charge). Moreover, the lemma shows how to calculate $\alpha_b(P)$ of a path P, where $b \in [0, B]$, using the largest gain of a vertex on P and $g(P)$, see Figure 11.

[18]Recall Definition B.1 which states that we don't drop charge at v_1.



(a)



(b)

Figure 11: Illustration of Lemma C.1. We start at v_1 with b charge. The subpath $v_i \dots v_j$ has the lowest gain $g(v_i \dots v_j) = \min_{i'<j'} g(v_{i'} \dots v_{j'})$. As depicted, $|g(v_i \dots v_j)| \leq B$. Moreover, every prefix $v_1 \dots v_t$ has gain $g(v_1 \dots v_t) \geq -b$ and therefore $\alpha_b(P) \neq -\infty$. The vertex of maximum gain is $v_{i^\star} = \operatorname{argmax}_{v_i} g(v_1 \dots v_i)$. If $b + g(v_1 \dots v_{i^\star}) \leq B$, see Figure (a), then $\alpha_b(v_1 \dots v_{i^\star}) = b + g(v_1 \dots v_{i^\star})$. Otherwise, see Figure (b), $\alpha_b(v_1 \dots v_{i^\star}) = B$. In both cases $\alpha_b(v_1 \dots v_k) = \alpha_b(v_1 \dots v_{i^\star}) + g(v_{i^\star} \dots v_k)$.

Lemma C.1. *Let $P = v_1 \dots v_k$ and $b \in [0, B]$. Then*

- *$\alpha_b(P) \geq 0$ if and only if for every $1 \leq j \leq k$ it holds that $g(v_1 \dots v_j) \geq -b$ and for every $1 \leq j_1 \leq j_2 \leq k$ it holds that $g(v_{j_1} \dots v_{j_2}) \geq -B$.*
- *If $\alpha_b(P) \geq 0$ then $\alpha_b(P) = \min\{B, b + g(v_1 \dots v_{i^\star})\} + g(v_{i^\star} \dots v_k)$, where $v_{i^\star} = \operatorname{argmax}_{v_i} g(v_1 \dots v_i)$.*

Proof. TOPROVE 6 □

The following lemma is used extensively in order to lower bound $\alpha_b(P)$, for a monotone path P and $b \in [0, B]$, by the gain of P.

Lemma C.2. *Let $P = v_1 \dots v_k$ be a monotone path with respect to a charge drop schedule C. Let $b \in [0, B]$.*

- *If P is descending with respect to C and $g^C(P) \geq -b$, then $\alpha_b(P) \geq b + g^C(P)$.*
- *If P is ascending with respect to C, then $\alpha_b(P) = \min\{B, b + g(P)\} \geq \min\{B, b + g^C(P)\}$. In particular, P is strongly traversable.*

Proof. TOPROVE 7 □

Let $P = v_1 \dots v_k$ be a negative arc-bounded path. The following lemma states that if the (negative) bounding arc of P has gain at least $-B$ then P us traversable.

Lemma C.3. *Let $P = v_1 \dots v_k$ be a negative arc-bounded path with respect to a charge drop schedule C. Let $b \in [0, B]$.*

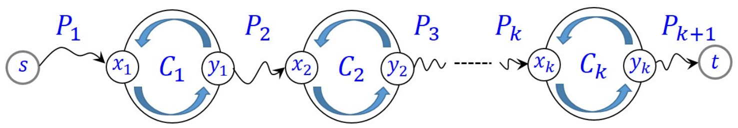


Figure 12: Generic structure of minimum energetic paths in the presence of negative cycles. If $\alpha_b(s,t) \geq -\infty$, then there is a minimum energetic path from s to t of the form shown, where $C_1, \ldots, C_k$ are simple negative cycles and (x_i, y_i) is an entry-exit pair on C_i, for $i = 1, 2, \ldots, k$. All entries $x_1, x_2, \ldots, x_k$ are distinct and all exits $y_1, y_2, \ldots, y_k$ are distinct. The paths $P_1, P_2, \ldots, P_{k+1}$ are simple but necessarily disjoint from the cycles $C_1, C_2, \ldots, C_k$.

- *If P is first arc-bounded with respect to C and $g(v_1v_2) \geq -b$, then $\alpha_b(P) \geq b + g^C(P)$.*
- *If P is last arc-bounded with respect to C and $g(v_{k-1}v_k) \geq -B$ and $g^C(P) \geq -b$, then $\alpha_b(P) \geq \min\{b + g^C(P), B + g(v_{k-1}v_k)\}$.*

Proof. TOPROVE 8 □

The following structural definition and lemma are from Dorfman et al. [5].

Definition C.4 (Entry-exit pairs [5]). *Let C be a positive gain cycle in $G = (V, A, g)$ and let B be the capacity of the battery. A pair of vertices (x, y) on C is an* entry-exit *pair of C if the car can start at x with an empty battery and eventually get to y, possibly after going several times around the cycle, with a full battery, i.e., with a charge of B.*

The following lemma characterise the structure of optimal paths, see Figure 12.

Lemma C.5 (Lemma 2.6 of [5]). *If there is a traversable path P from s to t in G, then there is a traversable path P' from s to t such that $\alpha_b(P') \geq \alpha_b(P)$, for every $b \in [0, B]$, where P' has the following form: either P' is simple, or there is a sequence $C_1, C_2, \ldots, C_k$ of simple positive gain cycles, where $k < n$, with entry-exit pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ on them, such that P' is composed of a simple path from s to x_1, followed by sufficiently many traversals of C_1 that end in y_1 with a full battery, followed by a simple path from y_1 to x_2, followed by sufficiently many traversals of C_2 that end in y_2 with a full battery, and so on, and finally a simple path from y_k to t. Furthermore, all entries $x_1, x_2, \ldots, x_k$ are distinct, and all exits $y_1, y_2, \ldots, y_k$ are distinct.*

D Overview of the Algorithm

The algorithm is composed of two stages. The goal of first stage, which is described in Appendix E, is to store information about shortcuts that correspond to simple paths. In the second stage, which is described in Appendix H, we use the stored information and compute $\alpha_B(s,t)$ for every $s, t \in V$.

D.1 Stage I

This stage performs $O\left(n^\alpha \log^2 n\right)$ iterations. To clarify the presentation we partition these iterations into $O\left(\log^2 n\right)$ outer-iterations, which are performed in *Compute-Shortcuts* (see Figure 13), where each of them calls the procedure *Update-Shortcuts*(M) which performs $O(n^\alpha)$ inner-iterations.

In each inner-iteration we take $M \in \mathbb{R}^{n \times n}$, our current table of shortcuts, and improve it several times. These improvements happen using two procedures *Short-Shortcuts* and *Long-Shortcuts*, which take M and return an improved table M'. Both procedures perform computations solely on G^M, which is the complete graph whose arc gains are defined according to M, i.e., $g(uv) = M[u][v]$ for every $u, v \in V$.

At inner-iteration i, we store the shortcuts (and more information) in a data structure D. The data structure is a union of 3 tables. Let $x, y, z \in V$ and let G^M be the current graph of interest. The values in D are defined with respect to G^M.

- $D[x][y]$ is the maximum gain of a monotone path from x to y we have encountered.[19]
- $D[xy][z]$ is the maximum gain of a $\bar{x}\underline{y}z$ path or a $\underline{x}\bar{y}z$ path.
- $D[x][yz]$ is the maximum gain of a $x\bar{y}\underline{z}$ path or a $x\underline{y}\bar{z}$ path.

We show in Corollary G.2 that these values also correspond to paths in G with at least as much gain. The following definition helps us to measure the quality of the values stored in D

Definition D.1. *Let $P = v_1 \dots v_k$ be a path in G^M. We say that the data structure D* dominates *P with respect to a charge drop schedule C if*

- *If P is v_1v_2-bounded with respect to C, then $D[v_1v_2][v_k] \geq g^C(P)$.*
- *If P is $v_{k-1}v_k$-bounded with respect to C, then $D[v_1][v_{k-1}v_k] \geq g^C(P)$.*
- *If P is monotone with respect to C, then $D[v_1][v_k] \geq g^C(P)$.*

If C is the zero schedule we just say that D dominates P.

Let M be the final table of shortcuts computed during Stage I. Theorem F.1, states that for any simple monotone path $P = v_0 \dots v_k$ in G, w.h.p., $M[v_0][v_k] \geq g(P)$. That is, the gain of the arc (v_0, v_k) in G^M is larger than $g(P)$. This theorem follows from Lemma F.2 which shows that if P is a monotone path from v to w in G^{M_i} where M_i is the shortcuts table at the beginning of outer-iteration i, then there exists a monotone path P' from v to w in $G^{M_{i+1}}$, where M_{i+1} is the shortcuts table at the beginning of outer-iteration $i+1$, such that $g(P') \geq g(P)$ and $|P'| = \left(1 - \Omega\left(\frac{1}{\log n}\right)\right)|P|$.

D.2 Stage II

We begin by utilizing the shortcuts obtained from Stage I and build an auxiliary graph $H = (V^0 \cup V^B, E(H))$, where $V^b = \{v^b \mid v \in V\}$ for $b = 0, B$ represents that we are at v with at least b charge. An arc $u^{b_1}v^{b_2} \in E(H)$ represents that $\alpha_{b_1}(u,v) \geq b_2$. We add to $E(H)$ arcs that we can easily deduce by the shortcuts of Stage I. We compute the transitive closure $H^\star$ of H that has even stronger relations. We prove in Theorem H.12, that for every $s, t \in V$ we have $\alpha_B(s,t) = B$ if and only if $s^B t^B \in E(H^\star)$. Recall the "cycle-hopping" structure of optimal cycles given by Lemma C.5. Let $s, t \in V$ and let P be an optimal path from s to t (i.e., $\alpha_B(P) = \alpha_B(s,t)$) that is structured as in Lemma C.5. Let $(x_1, y_1), \dots (x_k, y_k)$ be the entry-exit pairs as in Lemma C.5. By the discussion above, $s^B y_k^B \in E(H^\star)$, thus $H^\star$ allows us to skip all of the cycle-hoping and focus on the last path from y_k to t. This path is simple and we start traversing it with full charge. We prove in Lemma H.15 that $\alpha_B(y_k, t)$ can be derived from a value in D that correspond to a funnel in G^M, where M is taken from the last iteration of Stage I.

E Stage I - Algorithm for finding shortcuts

The goal of algorithm *Compute-Shortcuts*(G) (see Figure 13) is to find shortcuts corresponding to monotone simple paths in G. The algorithm proceeds in $\log^2 n$ outer-iterations. Each iteration is implemented using the procedure *Update-Shortcuts*(M), which gets the shortcuts table M of the previous iteration and computes new shortcuts, based on the graph defined by M. We claim (see Lemma F.2) that in each iteration, a monotone path P, consisting of shortcuts of the previous iteration, can be replaced by a shorter path Q (consisting of new shortcuts) of length shorter by a factor of $1 - \frac{c}{\log n}$, for some constant c.

[19]This value might be negative, or even $-\infty$. This is true also for the next entries.

```
Compute-Shortcuts($G = (V, A, g)$):
    $M \leftarrow ConstMarix(n, n, -\infty)$
    for $i = 1 \dots n$ do
        $M[i][i] \leftarrow 0$
        for $(i, j) \in E$ do
            $M[i][j] \leftarrow g(i, j)$
    for $t = 1 \dots \Theta(\log^2 n)$ do
        $M \leftarrow$ Update-Shortcuts($M$)
    return $M$
```

```
Update-Shortcuts($M$):
    $r \leftarrow \Theta(n^\alpha)$
    $M' \leftarrow M$
    for $i = 1 \dots r$ do
        $rand \sim U[0, 1]$
        if $rand < \log(n)/r$ :
            $M' \leftarrow \max\{M', $ Long-Shortcuts($M$)$\}$
        $M \leftarrow$ Short-Shortcuts($M$)
    return $\max\{M, M'\}$
```

Figure 13: Main procedure. Finds shortcuts corresponding to monotone simple paths. It combines many rounds of finding short shortcuts, with rare applications of finding shortcuts that correspond to longer paths ("long shortcuts")

The procedure *Update-Shortcuts*(M), which implements an outer-iteration, proceeds as follows. We perform $\tilde{O}(n^\alpha)$ rounds, which we view as inner-iterations, where $\alpha = 0.5$ is a constant that we set later. In each round, we call the procedure *Short-Shortcuts*(M) which finds all short shortcuts in G^M and updates M accordingly, see Figure 14. Also, with a small probability $p = \tilde{\Theta}\left(n^{-\alpha}\right)$ during a round, we also call the procedure *Long-Shortcuts*(M) which aims to find some k-shortcuts in G^M, where $k > 3$. This procedure also updates M.

Intuitively, given a monotone path P in G^M, *Update-Shortcuts*(M) aims to reduce its length by computing shortcuts in G^M that can replace monotone subpaths of P. Let $P_1, \dots, P_{n^\alpha}$ be the corresponding shortcutted versions of P with respect to the updated shortcuts tables $M_1, \dots, M_{n^\alpha}$. If we succeeded in reducing the length of P (say by a constant factor) by the $\tilde{O}(n^\alpha)$ applications of *Short-Shortcuts* (i.e. $|P_{n^\alpha}| \le c|P|$), then we achieved our goal. Otherwise, in most of the iterations we did not find many short shortcuts on P_i in G^{M_i}, and therefore P_i mostly consists of funnels (and some of them can be long). For this reason, with small probability (enough to "hit" such a round) we call *Long-Shortcuts* which finds some shortcuts that correspond to monotone paths that contain funnels. We prove in Lemma F.11, which is the central lemma of this paper, that these shortcuts are enough.

Each of these procedures computes a data structure D storing information about monotone and arc-bounded paths in G^M. At the end of each such call we update M with new shortcuts based on D.

The algorithm maintains the following invariant.

Invariant 1. *Let M be the shortcuts table of the current inner-iteration. The following holds throughout the inner-iteration:*

(A) If $D[xy][z] \neq -\infty$ then there is a traversable path $P = xy \dots z$ in G^M and a charge drop schedule C such that P is xy-bounded with respect to C and $g^C(P) = D[xy][z]$. We say that P is realizing $D[xy][z]$.

(B) If $D[x][yz] \neq -\infty$ then there is a traversable path $P = x \dots yz$ in G^M and a charge drop schedule C such that P is yz-bounded with respect to C and $g^C(P) \ge D[x][yz]$. We say that P is realizing $D[x][yz]$.

(C) If $D[x][y] \neq -\infty$ then there is a traversable path $P = x \dots y$ in G^M and a charge drop schedule C such that P is monotone with respect to C and $g^C(P) = D[x][y]$. Moreover, if $D[x][y] \ge 0$, then P is strongly traversable. We say that P is realizing $D[x][y]$.

The full details of the procedures *Short-Shortcuts* and *Long-Shortcuts* are explained in Appendices E.2 and E.3.6, respectively.

E.1 Initializing the data structure

At the beginning of *Short-Shortcuts*(M) and *Long-Shortcuts*(M), we get a shortcuts table M and initialize the data structure D with respect to G^M. This initialization creates trivial paths: For every $x, y \in V$ we set $D[x][y] = M[x][y]$ and $D[x][xy] = D[xy][y] = M[x][y]$.

E.2 Short Shortcuts

The goal of this procedure is to find shortcuts that dominate all (ascending/descending) monotone paths in G^M of length at most 3, see Figures 14 and 15. Finding 2-shortcuts is easy. We find them all by simply checking for every triplet $x, y, z \in V$ whether xyz is an ascending path in G^M, see Figure 15$(a2)$, and if not, we create a descending shortcut by dropping the right amount of charge, see Figure 15$(b1)$-$(b3)$. We classify monotone paths $xyaz$ of length 3 according to the eight possibilities for the *sign* of $M[x][y], M[y][a]$ and $M[a][z]$. In seven out of the eight cases we compute shortcuts that dominate these paths similarly to the computation of 2-shortcuts: For every $x, y, z \in V$ we concatenate the arc xy with the length 2 shortcut from y to z that were previously computed. We then check whether this results in an ascending shortcut, see Figure 15$(a1)$, and otherwise we perform charge drops to get a descending shortcut, see Figure 15$(c1) - (c6)$. This is done in *Trivial-Shortcuts*(M, D), see Figure 15. This procedure captures almost all of the possible monotone paths of length at most 3, except for three cases shown in Figure 14.

The three special cases of monotone paths $xyaz$ are the following. In all cases the signs of the arc gains alternate between positive and negative

Case 1: In this case the *sign* pattern of $M[x][y], M[y][a], M[a][z]$ is the same as in Figure 15$c(3)$. That is, $M[x][y], M[a][z] \geq 0$ and $M[y][a] \leq 0$. Here the path $xyaz$ is ascending and therefore we create an ascending shortcut from x to z, see Figure 14(a).

The last two cases are associated with the sign pattern $M[x][y], M[a][z] \leq 0$ and $M[y][a] \geq 0$.

Case 2: The path $xyaz$ satisfies $g_a \in [g_y, g_x]$. In this case either $g_z \leq g_y$, meaning that $xyaz$ is descending, or $g_z \in [g_y, g_a]$. In the latter case we can perform a charge drop at z to make $xyaz$ descending with respect to the appropriate schedule, see Figure 14(b).

Case 3: The path $xyaz$ satisfies $g_a > g_x (= 0)$. In this case, in order to make $xyaz$ descending, we perform a charge drop at a such that its gain after the drop is the same as the gain of x (which is zero). If after this charge drop z has larger gain than y, then we also perform a charge drop at z so that it matches the gain of y, see Figure 14(c).

The computation of these three cases of shortcuts is done in *Short-Shortcuts*(M) (see Figure 14) as follows. For every triplet $x, y, z \in V$ we do the following. In all cases we aim to compute shortcuts with gains as large as possible.

Assume $M[x][y] > 0$, we try to find shortcuts corresponding to Case 1. That is, we find $a \in V$ such that $xyaz$ is ascending and a satisfies $M[y][a] \leq 0$ and $M[a][z] \geq 0$, see Figure 14(a). The computation of $a \in V$ is done as follows. Among all nonpositive gain arcs ya whose gain in absolute value is smaller than the gain of xy, we want to find the one which maximized $M[y][a] + M[a][z]$ and $M[y][a] + M[a][z] \geq 0$. To make this search efficient we store all the pairs $(M[y][a], M[y][a] + M[a][z])$, for $a \in V$, in a $2D$ range tree T_{yz}. Creating such a range tree T_{yz} can be done in $O(n \log^2 n)$ time. Finally, our update for the triplet $x, y, z \in V$ will find amongst pairs in T_{yz} in which the first key $M[y][a]$ satisfies $M[y][a] \in [-M[x][y], 0]$, the pair in which its second key $M[y][a] + M[a][z]$ is maximal.[20] This is done in $O(\log^2 n)$ time both in the construction and initialization.

Assume $M[x][y] < 0$, finding shortcuts corresponding to Case 2 is done similarly to shortcuts corresponding to Case 1 by utilizing the range tree T_{yz}, see Figure 14(b). We find shortcuts corresponding

[20]It is enough to use a range tree in which the secondary structures are heaps rather than search trees because we only need to find the maximum in every secondary data structure. This saves a $\log n$ factor.

```
Short-Shortcuts($M$):
    $D \leftarrow Init\text{–}DS(M)$
    Trivial-Shortcuts($M, D$) // Adding to $D$ both 2-shortcuts and easy 3-shortcuts in $G^M$
    for $y, z \in V$ do // Creating $2D$ range trees for $yaz$ paths
        $T_{yz} \leftarrow RT(M[y][\cdot], M[y][\cdot] + M[\cdot][z])$
        $T'_{yz} \leftarrow RT(M[y][\cdot], M[\cdot][z])$
    for $x, y, z \in V$ do // 3-shortcuts
        if $M[x][y] \geq 0$ :
            $(-, k_2) \leftarrow T_{yz}.range(k_1 \in [-M[x][y], 0]).max\_k_2()$
                                        // largest gain at $z$ without going below $x$
            if $M[x][y] + k_2 \geq M[x][y]$ : // ascending shortcut
                $D[x][z] \leftarrow \max\{D[x][z], M[x][y] + k_2\}$
        if $M[x][y] \leq 0$ :
            $(-, k_2) \leftarrow T_{yz}.range(k_1 \in [0, |M[x][y]|]).max\_k_2()$
                                        // Largest gain at $z$ without going above $x$
            if $M[x][y] \geq M[x][y] + k_2 \geq -B$ : // descending shortcut
                $D[x][z] \leftarrow \max\{D[x][z], M[x][y] + k_2\}$
            if $M[x][y] + k_2 \geq M[x][y]$ : // descending shortcut, charge drop is needed at $z$
                $D[x][z] \leftarrow \max\{D[x][z], M[x][y]\}$
            $(-, M[a][z]) \leftarrow T'_{yz}.range(k_1 \in [|M[x][y]|, \infty)).max\_k_2()$
                                        // Largest gain of last arc while going above $x$
            $D[x][z] \leftarrow \max\{D[x][z], \min\{M[x][y], M[a][z]\}\}$        // Charge drop at $a$ and $z$
    return $D.shortcuts$
```

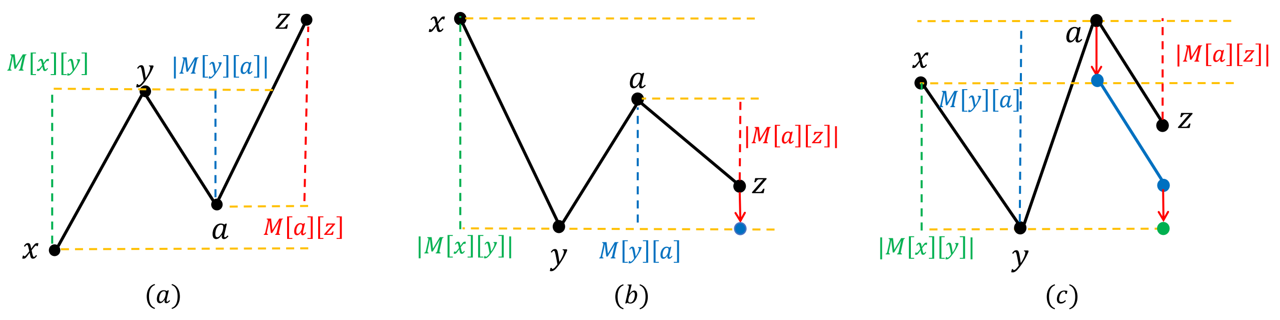


Figure 14: Hardest cases to compute 3-shortcuts. In these cases the signs of the arc gains alternate between positive and negative.

to Case 3 as follows. Among all $a \in V$ such that $M[y][a] \geq |M[x][y]|$, we find $a \in V$ such that $M[a][z]$ is maximized, see Figure 14(c). To make this search efficient we store all the pairs $(M[y][a], M[a][z])$, for $a \in V$, in a $2D$ range tree T'_{yz}, for every pair $y, z \in V$.

The pseudocode of *Short-Shortcuts*(M) is given in Figure 14. This pseudocode, and the pseudocodes of the algorithms in the next sections, use range trees as follows. Let K_1 and K_2 be arrays of length n. We denote by $RT(K_1[\cdot], K_2[\cdot])$ the operation of creating a $2D$ range tree with key pairs $(K_1[i], K_2[i])$, for $1 \leq i \leq n$.

Lemma E.1. *Let P be a monotone path in G^M of length $k \in \{2, 3\}$ with respect to a charge drop schedule C. Then at the end of Short-Shortcuts(M), D dominates P with respect to C.*

Proof. TOPROVE 9 $\square$

```
*Trivial-Shortcuts*$(M, D)$:
    $M_2, M_3 \leftarrow ConstMatrix(n, n, -\infty)$        // Matrices for shortcuts of paths of length 2 or 3
    for $x, y, z \in V$ do // 2-shortcuts
        if $M[x][y] \geq 0 \wedge M[y][z] \geq 0$ : // ascending shortcuts
            $M_2[x][z] \leftarrow \max\{M_2[x][z], M[x][y] + M[y][z]\}$
        else: // descending shortcuts with charge drop
            if $\min\{M[x][y], 0\} + \min\{M[y][z], 0\} \geq -B$ :
                $M_2[x][z] \leftarrow \max\{M_2[x][z], \min\{M[x][y], 0\} + \min\{M[y][z], 0\}\}$
    for $x, y, z \in V$ do // easy 3-shortcuts
        if $M[x][y] \geq 0 \wedge M_2[y][z] \geq 0$ : // ascending shortcuts
            $M_3[x][z] \leftarrow \max\{M_3[x][z], M[x][y] + M_2[y][z]\}$
        else: // descending shortcuts with charge drop
            if $\min\{M[x][y], 0\} + \min\{M_2[y][z], 0\} \geq -B$ :
                $M_2[x][z] \leftarrow \max\{M_2[x][z], \min\{M[x][y], 0\} + \min\{M_2[y][z], 0\}\}$
    for $x, z \in V$ do
        $D[x][z] \leftarrow \max\{D[x][z], M_2[x][z], M_3[x][z]\}$
```

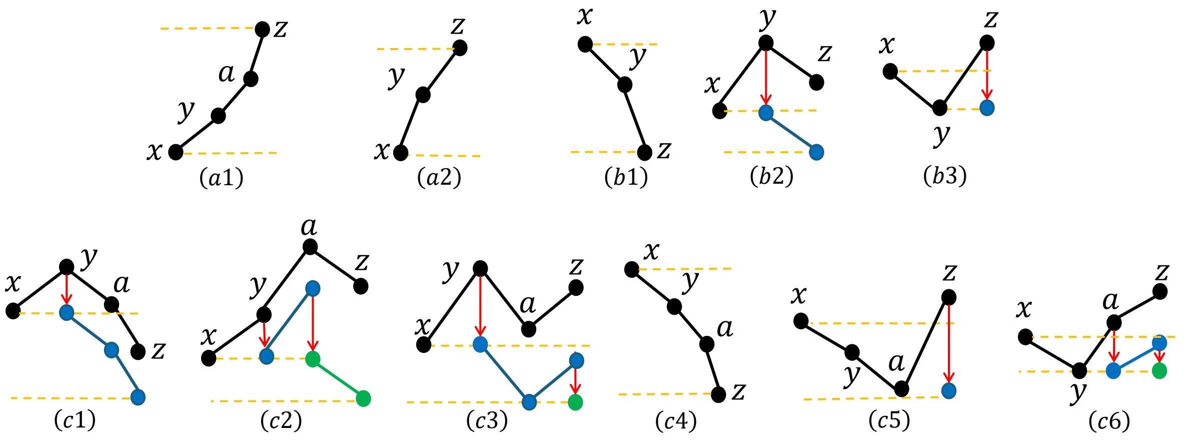


Figure 15: All cases of easy shortcuts, the dashed yellow lines represent the maximum and minimum gains in the paths (with respect to the charge drop schedule). The depicted charge drop schedules are optimal, i.e., the paths are descending with largest possible gain. Figures $(a1), (a2)$ are the only ascending shortcuts. Figures $(b1)$-$(b3)$ are descending 2-shortcuts with respect to a charge drop schedule that cancels every positive gain arc. Figures $(c1)$-$(c6)$ are descending 3-shortcuts. Note that the suffix yaz of these paths (except for $(c3)$) has the same schedule as in $(b1)$-$(b3)$. Case $(c3)$ is special since if z was higher, then $xyaz$ was ascending. This is handled in *Short-Shortcuts*(M).

Lemma E.2. *Procedure Trivial-Shortcuts*(M, D) *maintains Invariant 1(C).*

Proof. TOPROVE 10 □

Lemma E.3. *Procedure Short-Shortcuts*(M) *maintains Invariant 1(C).*

Proof. TOPROVE 11 □

E.3 Building Long Shortcuts

The procedure *Long-Shortcuts*(M) aims to find *long shortcuts* in G^M and update M accordingly. *Long Shortcuts* are shortcuts that correspond to monotone paths of length $k > 3$. We find such shortcuts by computing arc-bounded paths and then extending them by one arc into monotone paths (i.e shortcuts). We give the full description of *Long-Shortcuts*(M) in Appendix E.3.6. This algorithm uses several sub-algorithm which we list below and elaborate on in the next sections.

- *Breadth-Search*(M, D) : This procedure aims to discover arc-bounded paths that are longer than the ones stored in D. This is done by extending existing arc-bounded paths in D by one arc. This procedure performs updates of the form $D[xy][z] = \max\{D[xy][z], D[xy][a] + M[a][z]\}$. See Figure E.3.1.
- *Concatenate*(M, D, U, W, X): Given sets $U, W, X \subseteq V$, the procedure aims to discover longer arc-bounded paths than the ones stored in D by concatenating first-arc-bounded paths with first-arc-bounded paths and last-arc-bounded paths with last-arc-bounded paths. This procedure performs updates of the form $D[uv][x] = \max\{D[uv][x], D[uv][w] + D[wa][x]\}$, where $u \in U, w \in W, x \in X$. See Figure 17.
- *Compute-Funnels*(M) : This procedure returns a data structure D that dominates any simple path that is a funnel in G^M w.h.p. (see Lemma E.10). See Figure 18.
- *Concatenate-Opposite*(M, D, U, W, X): Given sets $U, W, X \subseteq V$, this procedure aims to discover longer arc-bounded paths than the ones stored in D by concatenating first-arc-bounded paths with last-arc-bounded paths. This procedure performs updates of the form $D[uv][x] = \max\{D[uv][x], D[uv][w] + D[w][ax]\}$, where $u \in U, w \in W, x \in X$. See Figure 19.
- *Arc-Bounded-To-Monotone*(M, D, T): Given a set $T \subseteq V$, this procedure considers every arc-bounded path in which the "bounding" arc contains a vertex of T. The goal of this procedure is to extend such a path by a single arc and get a monotone path. This is the procedure that computes the shortcuts for *Long-Shortcuts*(M, D).

The following is the relation between the different algorithms. Algorithm *Compute-Funnels*(M) Is achieved by applying *Breadth-Search* and *Concatenate* several times on a sampled set. Algorithms *Long-Shortcuts*(M) (see Figure 21) starts by applying *Compute-Funnels*(M), which returns a data structure D that, dominates every simple path that is a funnel in G^M w.h.p.. The algorithm then tries to elongate some sampled arc-bounded paths. This is done by consecutive applications of *Concatenate* and *Concatenate-Opposite*. Finally, *Long-Shortcuts*(M) calls *Arc-Bounded-To-Monotone* in order to transform the arc bounded path stored in D into monotone paths.

E.3.1 Breadth-Search

This procedure extend the length of arc-bounded paths dominated by D, by concatenating to them a single arc of larger gain. I.e., given $P = v_1 \dots v_k$, a v_1v_2-bounded path, *Breadth-Search*(M, D) scans all arcs xv_1 and checks if $xv_1 \dots v_k$ is xv_1-bounded path and if so, updates $D[xv_1][v_k]$. The implementation is as follows and its pseudocode is given in Figure E.3.1.

For every triplet $x, y, z \in V$, we update $D[xy][z]$ as follows. If $M[x][y] \geq 0$, we consider the values $D[ya][z]$, for all $a \in V$ such that $-M[x][y] \leq M[y][a] \leq 0$, and we concatenate xy to the path corresponding to $D[ya][z]$, which results in a xy-bounded path to z. That is, for every $y, z \in V$, we find $a \in V$, that maximizes $M[x][y] + D[ya][z]$ while satisfying $-M[x][y] \leq M[y][a] \leq 0$. To compute such $a \in V$, we store in a range tree FT_{yz} the pairs $(k_1, k_2) = (M[y][a], D[ya][z])$ for every $a \in V$. To update $D[xy][z]$, we search in FT_{yz} for the pair $(k_1, k_2) = (M[y][a], D[ya][z])$ with largest k_2 that satisfies $k_1 \in [-M[x][y], 0]$. We then assign $D[xy][z] = \max\{D[xy][z], M[x][y] + D[ya][z]\}$.

The case $M[x][y] \leq 0$ and the cases that P is last-arc-bounded are symmetric, See Figure E.3.1.

```
*Breadth-Search*$(D, M)$:
    for $a, b \in V$ do
        $FT_{ab} \leftarrow RT(M[a][\cdot], D[a\cdot][b])$        // Range tree of $\underline{a}\bar{w}b$ and $\bar{a}\underline{w}b$ paths
        $LT_{ab} \leftarrow RT(M[\cdot][b], D[a][\cdot b])$        // Range tree of $a\underline{w}\bar{b}$ and $a\bar{w}\underline{b}$ paths
    for $x, y, z \in V$ do
        if $M[x][y] \geq 0$ :
            $(-, D[ya][z]) \leftarrow FT_{yz}.range(k_1 \in [-M[x][y], 0]).max\_k_2()$
            $D[xy][z] \leftarrow \max\{D[xy][z], M[x][y] + D[ya][z]\}$        // We do this if $D[ya][z] \neq -\infty$
        else:
            $(-, D[ya][z]) \leftarrow FT_{yz}.range(k_1 \in [0, |M[x][y]|]).max\_k_2()$
            $D[xy][z] \leftarrow \max\{D[xy][z], M[x][y] + D[ya][z]\}$        // We do this if $D[ya][z] \neq -\infty$
        if $M[y][x] \geq 0$ :
            $(-, D[z][ay]) \leftarrow LT_{zy}.range(k_1 \in [-M[y][x], 0]).max\_k_2()$
            $D[z][yx] \leftarrow \max\{D[z][yx], D[z][ay] + M[y][x]\}$        // We do this if $D[z][ay] \neq -\infty$
        else:
            $(-, D[z][ay]) \leftarrow LT_{zy}.range(k_1 \in [0, |M[y][x]|]).max\_k_2()$
            $D[z][yx] \leftarrow \max\{D[z][yx], D[z][ay] + M[y][x]\}$        // We do this if $D[z][ay] \neq -\infty$
```

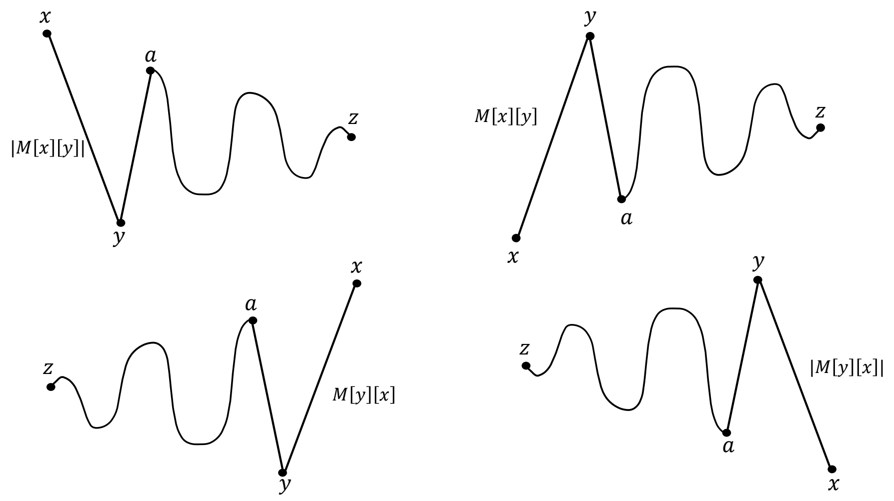


Figure 16: The four cases of *Breadth-Search*(M, D). On the top we concatenate the arc xy with a first-arc bounded path from y to z. On the bottom we concatenate a last-arc bounded path from z to y with the arc yx.

Lemma E.4. *Let $P = v_1 \dots v_k$ be an arc-bounded path in G^M. If D dominates P, then the following holds after Breadth-Search(D, M)*

- *If P is v_1v_2-bounded and $P' = v_0v_1v_2 \dots v_k$ is v_0v_1-bounded, then D dominates P'.*
- *If P is $v_{k-1}v_k$-bounded and $P' = v_1 \dots v_kv_{k+1}$ is v_kv_{k+1}-bounded, then D dominates P'.*

Proof. TOPROVE 12 $\square$

Since every funnel is arc-bounded, the following is a direct corollary of Lemma E.4.

Corollary E.5. *Assume that every funnel P in G^M of length at most k is dominated by D. Then after calling Breadth-Search(D, M), it holds that every funnel P in G^M of length at most $k+1$ is dominated by D.*

Lemma E.6. *Procedure Breadth-Search(M, D) maintains Invariants 1(A) and 1(B)*

Proof. TOPROVE 13 □

E.3.2 Concatenate first-arc bounded paths with first-arc bounded paths

In this procedure (see Figure 17) we are given 3 sets $U, W, X \subseteq V$. For every $u \in U, w \in W, x \in X$ and $v \in V$, we try to concatenate a $\bar{u}\underline{v}w$ path with some $\bar{w}\underline{a}x$ path, where $a \in V$. This gives a (hopefully new or improved gain) $\bar{u}\underline{v}x$ path. We also do the symmetric version: we try to concatenate a $x\bar{a}\underline{w}$ path to a $w\bar{v}\underline{u}$ path.

The choice of focusing on paths bounded by a arcs of negative gain was intentional. To emphasize the difficulty in concatenating paths bounded by arcs of positive gain, consider the following example.

Let P be a $\underline{u}\bar{v}w$ path, where $M[u][v] = 10$ and $g(P) = 5$. Let Q be a $\underline{w}\bar{a}x$ path, where $M[w][a] = 5$ and $g(Q) = 3$. Clearly $P \mid Q$ is uv-bounded with gain $g(P \mid Q) = 8$. However it may be the case where D dominates both P and Q and stores the values $D[uv][w] = 9$ and $D[wa][x] = 4$. But the concatenation of the paths, say P' and Q', realizing these values is not uv-bounded since the gain of $P' \mid Q'$ is $D[uv][w] + D[wa][x] = 13$ which is larger than $M[u][v]$. For arcs of negative gain if we replace P by a $\underline{u}\bar{v}w$ path P' with a larger gain then $P' \mid Q$ is always also uv-bounded. We could have addressed this problem by dropping charge at w (see Definition B.1) but we preferred to get our desired set of shortcuts without concatenating such paths at all.

In Appendix E.3.4 we show how to concatenate $\bar{u}\underline{v}w$ paths with $w\underline{a}\bar{x}$ paths. This requires a range tree and the ability to drop charges.

We distinguish the cases of concatenating first-arc-bounded paths and last-arc-bounded paths.

Concatenating $\underline{u}\bar{v}w$ and $\underline{w}\bar{a}x$: Consider the values D[uv][w] and $D[wa][x]$, where $a \in V$ satisfies $M[w][a] \leq 0$. It follows from Lemma E.7 and Lemma E.8 that the concatenation of the paths realizing these values is a uv-bounded path if and only if $|M[w][a]| \leq D[uv][w] - M[u][v]$. See Figure 17.

Therefore we update $D[uv][x]$ as follows. We find an $a \in V$ that maximises $D[uv][w] + D[wa][x]$ while satisfying $|M[w][a]| \leq D[uv][w] - M[u][v]$. This is done by storing, for every pair $w \in W, x \in X$, a Range tree of first-arc-bounded paths FT_{wx} containing the pairs $(k_1, k_2) = (M[w][a], D[wa][x])$, for every $a \in V$. We then find the pair $(k_1, k_2) = (M[w][a], D[wa][x])$ with largest k_2 that satisfies $k_1 \in [-(D[uv][w] - M[u][v]), 0]$. We then perform the update $D[uv][x] = \max\{D[uv][x], D[uv][w] + D[wa][x]\}$.

Concatenating $x\bar{a}\underline{w}$ and $w\bar{v}\underline{u}$: This case is handled symmetrically. We perform an update of the form $D[x][vu] = \max\{D[x][vu], D[x][aw] + D[w][uv]\}$, see Figure 17.

The following lemma proves that after running algorithm *Concatenate*(M, D), the concatenation of two arc-bounded paths P, Q that match the description above and were dominated by D before executing *Concatenate*(M, D), is dominated by D after this execution.

Lemma E.7. *Let $U, W, X \subseteq V$ and let $u \in U, w \in W, x \in X$ and $v \in V$. Let P_1 and P_2 be paths in G^M that are dominated by D. Assume one of the following holds*

- *P_1 is a $\bar{u}\underline{v}w$ path, P_2 is a $\bar{w}\underline{a}x$ path, and $P = P_1 \mid P_2$ is a $\bar{u}\underline{v}x$ path.*
- *P_1 is a $x\bar{a}\underline{w}$ path, P_2 is a $w\bar{v}\underline{u}$ path, and $P = P_1 \mid P_2$ is a $x\bar{v}\underline{u}$ path.*

Then, after Concatenate(M, D, U, W, X), D dominates P.

```
$Concatenate(M, D, U, W, X)$:
    for $(w, x) \in W \times X$ do
        $FT_{wx} \leftarrow RT(M[w][\cdot], D[w\cdot][x])$            // Range tree of $\bar{w}\underline{a}x$ paths
        $LT_{xw} \leftarrow RT(M[\cdot][w], D[x][\cdot w])$            // Range tree of $x\bar{a}\underline{w}$ paths
    for $(u, v, w, x) \in U \times V \times W \times X$ do
        if $M[u][v] < 0$ :
            $(-, D[wa][x]) \leftarrow FT_{wx}.range(k_1 \in [-(D[uv][w] - M[u][v]), 0]).max\_k_2()$
            $D[uv][x] \leftarrow \max\{D[uv][x], D[uv][w] + D[wa][x]\}$
        if $M[v][u] < 0$ :
            $(-, D[x][aw]) \leftarrow LT_{xw}.range(k_1 \in [-(D[w][vu] - M[v][u]), 0]).max\_k_2()$
            $D[x][vu] \leftarrow \max\{D[x][vu], D[x][aw] + D[w][vu]\}$
```

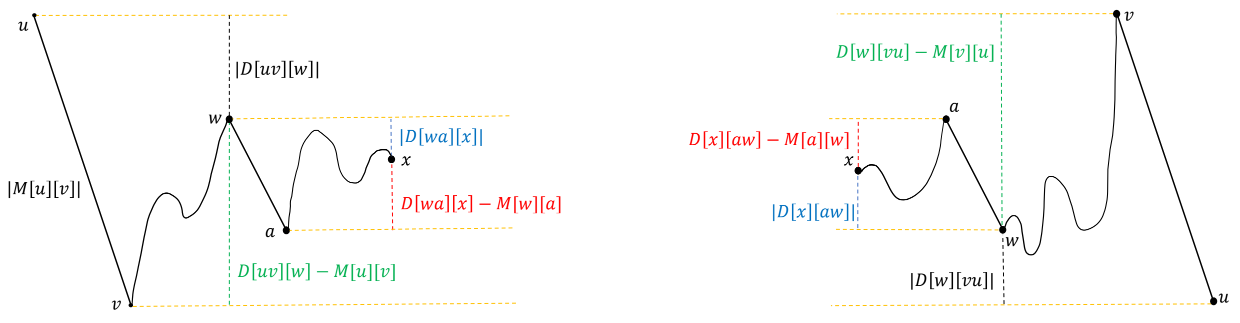


Figure 17: On the left: a concatenation of two $\bar{A}\underline{B}C$ paths.

Proof. TOPROVE 14 ☐

Lemma E.8. *Procedure Concatenate(M, D, U, W, X) maintains Invariants 1(A) and 1(B).*

Proof. TOPROVE 15 ☐

E.3.3 Dominating Funnels

This procedure returns a data structure D such that every funnel, that is a simple path in G^M, is dominated by D w.h.p.. This is done in 4 steps. Let $s = \tilde{O}(n^\beta)$, where $\beta = 2/3$. The first step is to compute bounded paths that dominate funnels of length n/s. This is done by running *Breadth-Search*(D, M) n/s times. Correctness of this step follows from Corollary E.5.

In the second step, we sample a set S of $\Theta(s \log n)$ vertices. For every triplet $s_1, s_2, s_3 \in S$ we try to concatenate a $\bar{s}_1\underline{a}_1 s_2$ path with a $\bar{s}_2\underline{a}_2 s_3$ path, where $a_1, a_2 \in V$. We also concatenate the symmetric paths: a $s_3\bar{a}_1\underline{s}_2$ path with a $s_2\bar{a}_2\underline{s}_3$ path. This is done by applying *Concatenate*(D, S, S, S) $\log n$ times, see Appendix E.3.2. Each of these $\log n$ iterations multiplies the length of the funnels between vertices of S that D dominates. We show that after the second step, D dominates all funnels that are simple paths. that start and end at vertices from S. Lemma E.9 proves the correctness of this step.

In the third step we call *Concatenate*(D, S, S, V), which for every $s_1, s_2 \in S$ and $v \in V$ concatenates $\bar{s}_1\underline{a}_1 s_2$ paths with $\bar{s}_2\underline{a}_2 v$ paths, where $a_1, a_2 \in V$. We also concatenate the symmetric paths: $v\bar{a}_1\underline{s}_2$ paths with $s_2\bar{a}_2\underline{s}_1$ paths. We show that after the third step, D dominates every simple funnel that is a $\bar{s}\underline{u}v$ path or a $v\bar{u}\underline{s}$ path, where $s \in S$ and $u, v \in V$. That is a funnel that starts with a sampled vertex and ends at an arbitrary vertex or a funnel that ends with a sampled vertex and starts at an arbitrary

```
Compute-Funnels($M$):
    $D \leftarrow$ Init−DS($M$)
    $s \leftarrow \Theta(n^{\beta})$
    for $i = 1, \ldots, n/s$ do                                  // Finding funnels of length $n/s$
        Breadth-Search($M, D$)
    $S \leftarrow$ Sample($V, p = \log n \cdot s/n$)                     // Each vertex is sampled i.i.d
    for iteration $= 1 \ldots \log n$ do
        Concatenate($M, D, S, S, S$)                  // Dominate funnels between sampled vertices
    Concatenate($M, D, S, S, V$)             // Compute suffixes of funnels (from sampled vertices)
    for $i = 1, \ldots, n/s$ do                                            // Fully compute funnels
        Breadth-Search($M, D$)
    return $D$
```

Figure 18: After this procedure every funnel P in G^M is dominated by D w.h.p.

vertex. This happens since each such funnel that ends at a vertex v contains w.h.p. a sampled vertex s, such that the funnel from s to v starts with a negative gain arc and is of length at most n/s.

Finally, in the fourth step we run *Breadth-Search*(D, M) again n/s times. This extends w.h.p. the funnels that we cover to include all simple funnels of linear length (that start at any vertex). Lemma E.10 proves the correctness of this entire procedure.

Lemma E.9. *Let $P = v_1 \ldots v_k$ be a funnel which is negative arc-bounded and let S be the set sampled by the procedure Compute-Funnels. Assume $v_1, v_k \in S$, then w.h.p. after applying Concatenate(M, D, S, S, S) $\log n$ times in Compute-Funnels(M), D dominates P.*

Proof. TOPROVE 16 □

Lemma E.10. *Let P be a funnel of length $|P| = O(n)$. After a call to Compute-Funnels(M), D dominates P w.h.p.*

Proof. TOPROVE 17 □

E.3.4 Concatenating first-arc-bounded paths with last-arc-bounded paths

Similarly to *Concatenate*(M, D), in *Concatenate-Opposite*(M, D) we are given 3 sets $U, W, X \subseteq V$. For every $u \in U, w \in W, x \in X$ and $v \in V$, we try to create a $\bar{u}\underline{v}x$ path by concatenating a $\bar{u}\underline{v}w$ path with a $w\underline{a}\bar{x}$ path, where we optimize over the choices of $a \in V$, see Figure 19. We also do the symmetric computation: we try to create a $x\bar{v}\underline{u}$ path by concatenating a $\underline{x}\bar{a}w$ path with a $w\bar{v}\underline{u}$ path. Notice that in either case the new path that we create is negative arc-bounded.

We now elaborate on the case corresponding to concatenating $\bar{u}\underline{v}w$ path with a $w\underline{a}\bar{x}$ path. Assume $M[u][v] < 0$, we update $D[uv][w]$ as follows. We consider the values $D[w][ax]$, for every $a \in V$ that satisfies $M[a][x] > 0$ and $M[a][x] - D[w][ax] \leq D[uv][w] - M[u][v]$. The latter condition guarantees that the gain of a is larger than the gain of v with respect to the concatenation of the paths realizing $D[uv][w]$ and $D[w][ax]$, see Figure 19. We distinguish between the following two cases.

Case 1: $\boldsymbol{M[a][x] - D[w][ax] \leq D[uv][w] - M[u][v]}$ and $\boldsymbol{D[w][ax] \leq |D[uv][w]|}$: This case corresponds to Figure 19(a). The first condition says that the gain of the concatenated path never goes below the gain of v (i.e $g_v = M[u][v] < 0$) and the second condition says that the gain of the concatenated path never exceeds the gain of u (i.e., $g_u = 0$). In this case we claim that the paths realizing $D[uv][w]$ and $D[w][ax]$ can be concatenated into a $\bar{u}\underline{v}x$ path of gain $D[uv][w] + D[w][ax]$. We find such an $a \in V$ with largest $D[w][ax]$ and perform the update $D[uv][x] = \max\{D[uv][x], D[uv][w] + D[w][ax]\}$. To find the best $a \in V$, we store for every $w \in W$ and $x \in X$ the pairs $(M[a][x] -$

$D[w][ax], D[w][ax])$, for $a \in V$ satisfying $M[x][a] > 0$, in a *Range Tree* LRT_{wx} of last-arc-bounded paths. We then perform a search in LRT_{wx} for a pair (k_1, k_2) with $k_1 \leq D[uv][w] - M[u][v]$ and largest k_2 that satisfies $k_2 \leq |D[uv][w]|$. This operation takes $O(\log^2 n)$ time.

Case 2: $D[w][ax] \geq |D[uv][w]|$ and $M[a][x] \leq |M[u][v]|$. This case corresponds to Figure 19(b). The first condition implies that the gain at x is larger than the gain at u. Note that the condition from Case 1 $M[a][x] - D[w][ax] \leq D[uv][w] - M[u][v]$ can be derived from the two conditions. In this case we set $D[uv][x] = 0$. To justify this assignment we argue that there is a path P and an associated charge drop schedule C such that P is a $\bar{u}\underline{v}x$ path with respect to C and $g^C(P) = 0$. Let (P_1, C_1) and (P_2, C_2) be the paths and charge drop schedules realizing $D[uv][w]$ and $D[w][ax]$, respectively. Let $P = P_1 \mid P_2$. We define a charge drop schedule C for P as follows: Let d_w be the last charge drop in C_1 associated with w. We get C by concatenating C_1 and C_2 and changing d_w to be equal to $d_w + g^{C_1}(P_1) + g^{C_2}(P_2)$.

We claim that P is uv-bounded with respect to C and $g^C(P) = 0$. The latter is clear since

$$g^C(P) = g^{C_1}(P_1) - (g^{C_1}(P_1) + g^{C_2}(P_2)) + g^{C_2}(P_2) = 0.$$

Lemma E.12 shows that P is uv-bounded with respect to C.

We discover whether there exists a vertex $a \in V$ for which we should apply this case as follows. For every $w \in W$ and $x \in X$, we store the pairs $(D[w][ax], M[a][x])$, for $a \in V$ satisfying $M[x][a] > 0$, in a 2-dimensional *Range Tree* LRT'_{wx} of values realized by $w\underline{a}\bar{x}$ paths. We then perform a search in LRT'_{wx} for a pair (k_1, k_2) with $k_1 \geq |D[uv][w]|$ and $k_2 \leq |M[u][v]|$. This operation is done in $O(\log^2 n)$ time. If we find such a pair, we apply this case and set $D[uv][x] = 0$.

The symmetric version, i.e., concatenating a $\underline{x}\bar{a}w$ path with a $w\bar{v}\underline{u}$ path, is as done analogously, see Figure 19. We search for $a \in V$, such that $M[x][a] > 0$ and one of the following cases is satisfied:

Case 3: $M[x][a] - D[xa][w] \leq D[w][vu] - M[v][u]$ and $D[xa][w] \leq |D[w][vu]|$: This case corresponds to Figure 19(c). Similarly to Case 1, in this case we can concatenate the paths realizing $D[xa][w]$ and $D[w][vu]$. we find a that maximize $D[xa][w]$ and perform the update $D[x][vu] = \max\{D[x][vu], D[xa][w] + D[w][vu]\}$.

Case 4: $D[xa][w] \geq |D[w][vu]|$ and $M[x][a] \leq |M[v][u]|$. This case corresponds to Figure 19(d). Similarly to Case 4, in this case in order to concatenate the paths realizing $D[xa][w]$ and $D[w][vu]$ we have to perform a charge drop at w. This will give us a $x\bar{v}\underline{u}$ path of gain 0 (with respect to some charge drop schedule), this is the best we can hope for in a negative arc-bounded path. We search if such an $a \in V$ exists using a Range tree FRT'_{xw}. If so, we perform the update $D[x][vu] = 0$.

Lemma E.11. *Let P and Q = be paths in G^M that are dominated by D. Let $U, W, X \subseteq V$ and let $u \in U, w \in W, x \in$ and $v \in V$. If one of the following holds*

- *P is a $\bar{u}\underline{v}w$ path, Q is a $w\underline{a}\bar{x}$ path, and $P \mid Q$ is a $\bar{u}\underline{v}x$ path.*
- *P is a $\underline{x}\bar{a}w$ path, Q is a $w\bar{v}\underline{u}$ path, and $P \mid Q$ is a $x\bar{v}\underline{u}$ path.*

then after Concatenate-Opposite(M, D, U, W, X), D dominates $P \mid Q$.

Proof. TOPROVE 18 □

Lemma E.12. *Procedure Concatenate-Opposite(M, D, U, W, X) maintains Invariants 1(A) and 1(B).*

Proof. TOPROVE 19 □

```
$Concatenate\text{-}Opposite(M, D, U, W, X)$:
    for $(w, x) \in W \times X$ do
        $LRT_{wx} \leftarrow RT()$                                  // $2D$-Range tree of $w\underline{a}\bar{x}$ paths
        for $a \in V$ $s.t$ $M[a][x] \geq 0$ do
            $LRT_{wx}.insert(M[a][x] - D[w][ax], D[w][ax])$
        $LRT'_{wx} \leftarrow RT()$                                 // $2D$-Range Tree of $w\underline{a}\bar{x}$ paths
        for $a \in V$ $s.t$ $M[a][x] \geq 0$ do
            $LRT'_{wx}.insert(D[w][ax], M[a][x])$
        $FRT_{xw} \leftarrow RT()$                                  // $2D$-Range tree of $\underline{x}\bar{a}w$ paths
        for $a \in V$ $s.t$ $M[x][a] \geq 0$ do
            $FRT_{xw}.insert(M[x][a] - D[xa][w], D[xa][w])$
        $FRT'_{xw} \leftarrow RT()$                                 // $2D$-Range tree of $\underline{x}\bar{a}w$ paths
        for $a \in V$ $s.t$ $M[x][a] \geq 0$ do
            $FRT'_{xw}.insert(D[xa][w], M[x][a])$
    for $(u, v, w, x) \in U \times V \times W \times X$ do
        if $M[u][v] < 0$ : // Trying to create a $\bar{u}\underline{v}x$ path
            $(-, D[w][ax]) \leftarrow LRT_{wx}.range(k_1 \leq D[uv][w] - M[u][v], k_2 \leq |D[uv][w]|).max\_k_2()$
            $D[uv][x] \leftarrow \max\{D[uv][x], D[uv][w] + D[w][ax]\}$
            $bool \leftarrow LRT'_{wx}.find(k_1 \geq |D[uv][w]|, k_2 \leq |M[u][v]|)$
            if $bool$ :
                $D[uv][x] \leftarrow 0$
        if $M[v][u] < 0$ : // Trying to create a $x\bar{v}\underline{u}$ path
            $(-, D[xa][w]) \leftarrow FRT_{xw}.range(k_1 \leq D[w][vu] - M[v][u], k_2 \leq |D[w][vu]|).max\_k_2()$
            $D[x][vu] \leftarrow \max\{D[x][vu], D[xa][w] + D[w][vu]\}$
            $bool \leftarrow FRT'_{xw}.find(k_1 \geq |D[w][vu]|, k_2 \leq |M[v][u]|)$
            if $bool$ :
                $D[x][vu] \leftarrow 0$
```

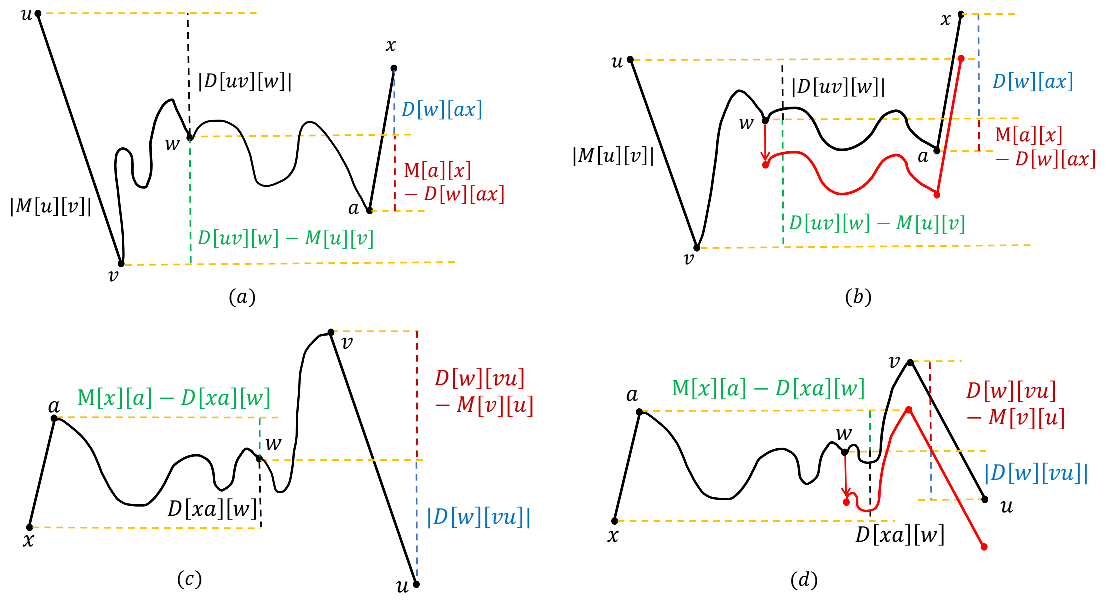


Figure 19: Concatenating arc-bounded paths, one is first-arc-bounded and the other is last-arc-bounded.

E.3.5 Build monotone paths from arc-bounded paths

In this procedure (see Figure 20) we are given a set $T \subseteq V$. For every $u \in T$ and $v, w, x \in V$, we try to concatenate a $\bar{u}\underline{v}x$ path with the arc xy in order to either get a descending path from u to y or to get an ascending path from v to y. We also do the opposite: we try to concatenate the arc yx with a $x\bar{v}\underline{u}$ path in G^M in order to either get a descending path from y to u or to get an ascending path from y to v.

The concatenation of $\bar{u}\underline{v}x$ paths with the arc xy is done as follows. We distinguish between the following cases.

Case 1: $\boldsymbol{-B \leq D[uv][x] + M[x][y] \leq M[u][v]}$. This case corresponds to a concatenation of a $\bar{u}\underline{v}x$ path with the arc xy that results in a descending path from u to u. The algorithm sets $D[u][y] = \max\{D[u][y], D[uv][x] + M[x][y]\}$.

Case 2: $\boldsymbol{D[uv][x] + M[x][y] \geq M[u][v]}$ This case means that after the concatenation, the gain at y is at least the gain at v. In order to make the path descending, we perform a charge drop at y such that the gain at y matches the gain of v, resulting in a descending path. The algorithm sets $D[u][y] = \max\{D[u][y], M[u][v]\}$.

Case 3: $\boldsymbol{D[uv][x] + M[x][y] \geq 0}$ This case means that after the concatenation, the gain at y is at least the gain at u (which is 0). This means that y has the maximum gain in the concatenated path. Since P is uv-bounded, v has the minimum gain in the concatenated path. Therefore the sub path from v to y is ascending. The algorithm sets $D[u][y] = \max\{D[u][y], -M[uv] + D[uv][x] + M[x][y]\}$.

The procedure performs similar computations when it concatenates the arc yx with a $x\bar{v}\underline{u}$ paths.

The following lemma states that if D dominates a first-arc uv-bounded path P, where $u \in T$, that can be extended by an arc xy and result in a monotone path P' (that starts either at u or v), then after *Arc-Bounded-To-Monotone*(M, D, T), D dominates P'. The lemma also proves a similar result for last-arc bounded paths.

Lemma E.13. *Let P be an arc-bounded path in G^M and assume that D dominates P. Denote by $\tilde{P}$ the subpaths of P that excludes the bounding arc of P (either the first arc or the last arc). Then the following holds after Arc-Bounded-To-Monotone(M, D, T)*

1. *Assume P is a $\bar{u}\underline{v}x$ bounded and $P' = P \mid y$ is descending such that $g(P') \geq -B$. If $u \in T$, then D dominates P'.*
2. *Assume P is $\bar{u}\underline{v}x$-bounded and $P' = \tilde{P} \mid y$ is ascending. If $u \in T$, then D dominates P'.*
3. *Assume P is a $\bar{u}\underline{v}x$ bounded. If $u \in T$, then $D[u][x] \geq M[u][v]$.*
4. *Assume P is a $x\bar{v}\underline{u}$ path and $P' = y \mid P$ is descending such that $g(P') \geq -B$. If $u \in T$, then D dominates P'.*
5. *Assume P is a $x\bar{v}\underline{u}$ path and $P' = y \mid \tilde{P}$ is ascending. If $u \in T$, then D dominates P'.*
6. *Assume P is a $u\bar{v}\underline{x}$ bounded. If $x \in T$, then $D[u][x] \geq M[v][x]$.*

Proof. TOPROVE 20 □

Lemma E.14. *Procedure Arc-Bounded-To-Monotone(M, D, T) maintains Invariant 1(A).*

Proof. TOPROVE 21 □

```
*Arc-Bounded-To-Monotone*$(M, D, T)$:
    for $(u, v, x, y) \in T \times V^3$ do
        if $M[u][v] \leq 0$ : // first-arc bounded paths
            if $-B \leq D[uv][x] + M[x][y] \leq M[u][v]$ : // descending shortcuts
                $D[u][y] = \max\{D[u][y], D[uv][x] + M[x][y]\}$
            if $D[uv][x] + M[x][y] \geq M[u][v]$ : // descending shortcuts
                $D[u][y] = \max\{D[u][y], M[u][v]\}$            // Drop charge at $y$ to be descending.
            if $D[uv][x] + M[x][y] \geq 0$ : // ascending shortcuts
                $D[v][y] = \max\{D[v][y], -M[u][v] + D[uv][x] + M[x][y]\}$
                                                        // Note: Ascending path starts at $v$.
        if $M[v][u] \leq 0$ : // last-arc bounded paths
            if $-B \leq M[y][x] + D[x][vu] \leq M[v][u]$ : // descending shortcuts
                $D[y][u] = \max\{D[y][u], M[y][x] + D[x][vu]\}$
            if $M[y][x] + D[x][vu] \geq M[v][u]$ : // descending shortcuts
                $D[y][u] = \max\{D[y][u], M[v][u]\}$            // Charge drop at $x$ to make $g^C_y = g^C_v$.
            if $M[y][x] + D[x][vu] \geq 0$ : // ascending shortcuts
                $D[y][v] = \max\{D[y][v], M[y][x] + D[x][vu] - M[v][u]\}$
                                                        // Note: Ascending path ends at $v$.
```

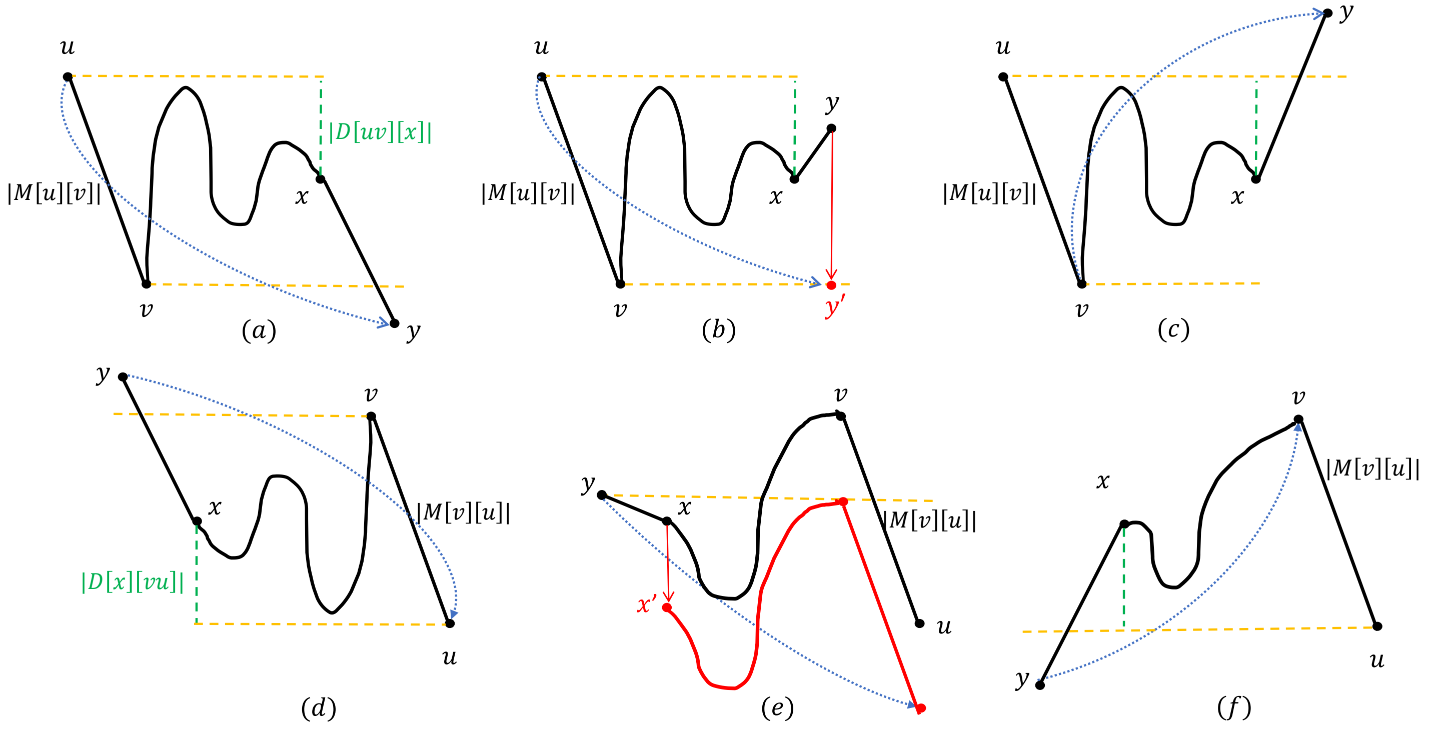


Figure 20: The six cases of the Algorithm *Arc-Bounded-To-Monotone*(M, D, T). Blue dotted arrows are new shortcuts. Red downwards vertical arrows represent charge drop at a vertex, this affects the gain of all subsequent vertices.

```
Long-Shortcuts(M):
    D ← Compute-Funnels(M)
    T ← ∅                                          // All vertices sampled for creating shortcuts
    for $i = 1 \ldots \log\left(n^{1-\alpha}\log^2 n\right)$ do
        $s_i \leftarrow \Theta(\frac{\log^2(n)}{2^i \cdot n^\alpha})$          // new sampling probability
        $T_i \leftarrow Sample(V, p = s_i)$
        repeat $2^i$ times
            Concatenate-Opposite($M, D, T_i, V, V$)     // Skip funnels of opposite direction
            Concatenate($M, D, T_i, V, V$)              // Skip funnels of the same direction
        $T \leftarrow T \cup T_i$
    Arc-Bounded-To-Monotone($M, D, T$)
    return D.shortcuts
```

Figure 21: Procedure *Long-Shortcuts*. We sample vertices and compute arc bounded paths in which those vertices are end points. The lower the sampling probability, the further we extend our search.

E.3.6 Long Shortcuts

This procedure aims to find "long shortcuts" in G^M. These are shortcuts that correspond to monotone paths of length $k > 3$. We find such shortcuts by computing (long) arc bounded paths and then extending them by one arc into monotone paths (i.e shortcuts) using *Arc-Bounded-To-Monotone*.

The procedure (See Figure 21) starts by running *Compute-Funnels*(M) in order get a data structure D that dominates each funnel in G^M w.h.p. The procedure *Long-Shortcuts*(M) samples sets T_i of size $\Theta(\frac{\log^2(n)\cdot\kappa}{2^i})$, for every $1 \le i \le \log n$,[21] where $\kappa = \Theta(n^{1-\alpha})$. In Appendix F κ would be a bound on the number of funnels which are maximal with respect to inclusion in a studied path P. For every $u \in T_i$, we concatenate 2^i times arc bounded paths starting at u (uw-bounded) with other arc bounded paths. This is done using the two concatenation procedures *Concatenate*(M, D) and *Concatenate-Opposite*(M, D). Intuitively, each such concatenation extends the reach of a uv-bounded path P ($u \in T_i$) by an additional funnel.

For example, if P is first-arc bounded, then the procedure *Concatenate-Opposite*(M, D) is used in order to concatenate P with last-arc bounded funnel and *Concatenate*(M, D) is used in order to concatenate P with first-arc bounded funnel.

Finally, after computing these arc bounded paths, we try to extend them by one arc to get new shortcuts. We do so by running *Arc-Bounded-To-Monotone*(M, D, T), where $T = \cup_i T_i$ is the set of all sampled vertices.

F Stage I Correctness

In this appendix we prove the main theorem of our shortcutting algorithm.

Theorem F.1. *Let $P = v_1 \ldots v_k$ be a monotone simple path in G. Let M be the shortcuts returned from Compute-Shortcuts(G). Then w.h.p. $M[v_1][v_k] \ge g(P)$.*

Theorem F.1 follows from the following lemma.

Lemma F.2. *Let $P = v_1 \ldots v_k$ be a monotone simple path in G^M with respect to a charge drop schedule C. Let M' be the shortcuts table after running Update-Shortcuts(M). If $|P| \le n^\alpha$, then*

[21] Recall the intuition from the Technical review (Section 2.2), the algorithm interpolates between two extreme cases, see Figure 5

$M'[v_1][v_k] \geq g^C(P)$. If $|P| > n^\alpha$, then w.h.p. there is a monotone path P', with respect to a charge drop schedule C', from v_1 to v_k in $G^{M'}$ that satisfies $g^{C'}(P') \geq g^C(P)$ and $|P'| \leq (1 - 1/\Omega(\log n)) \cdot |P|$.

Before proving Lemma F.2, we need to introduce the concept of funnel decomposition.

F.1 Funnel Decomposition

A *funnel decomposition* of a path $P = e_1 \dots e_k$ in G^M is a partition of P into subpaths $F_1, \dots, F_t$ which are funnels that are maximal with respect to inclusion. More precisely, the funnel decomposition of P is defined by the following process. We define $F_1 = e_1 \dots e_r$, where $1 \leq r \leq k$, to be the maximal funnel in P that contains e_1. Assume we have constructed $F_1, F_2, \dots, F_s$ and denote $F_s = e_\ell \dots e_r$. If $\cup_{i=1}^s F_i \neq P$, then we define $F_{s+1} = e_{\ell'} \dots e_{r'}$ as the maximal funnel in P with largest r' that contains e_{r+1}.[22] In particular $\ell' > \ell$. Since every arc is a funnel, it is clear that the funnel decomposition is well defined.

The following lemma proves structural properties on the funnel decomposition. The lemma states that every two different funnels that are maximal can intersect by at most two arcs. In particular, every two consecutive funnels in the funnel decomposition overlap by at most two consecutive arcs.

Lemma F.3. *Let $P = e_1 \dots e_k$ be a path in $G = (V, A, c)$. Let $F_1 = e_a \dots e_b$ and $F_2 = e_c \dots e_d$ be two different funnels in P which are maximal with respect to inclusion. If $a < c$ then $c \geq b - 1$. Moreover, if $c = b - 1$ then $g(e_{b-1}) = -g(e_b)$*

Proof. TOPROVE 22 □

F.2 Proof of Lemma F.2

We present the road map of the proof of Lemma F.2. Let $u, v \in V$ and let P be an ascending path[23] from u to v in G^M. Let $M_1, \dots, M_r$ be the shortcuts tables resulted after each of the $r = n^\alpha$ applications of *Short-Shortcuts* during the iterations of *Compute-Shortcuts*. During these iterations we called *Short-Shortcuts* n^α times and *Long-Shortcuts* $\tilde{\Theta}(1)$ times in expectation. Let P_i be the shortest ascending path from u to v in G^{M_i} of gain larger than the gain of P in G^M. Since the shortcuts tables M_i keep increasing their gains it follows that $|P_i|$ is decreasing with $i = 1, \dots, r$. Let us focus only on the calls of *Short-Shortcuts*(M_i) for $i = 1, \dots, r$. If after running *Short-Shortcuts* n^α times, the length of P_{n^α} is not smaller by a constant factor than the length of P, say $P_{n^\alpha} \geq 0.9|P|$, it follows that in **half of the calls** to *Short-Shortcuts*(M_i) we have $|P_i| - |P_{i+1}| \leq 0.2|P|/n^\alpha$.

Lets focus on an iteration i such that $|P_i| - |P_{i+1}| \leq 0.2|P|/n^\alpha$. This means that in P_i there are at most $0.2|P|/n^\alpha$ short shortcuts. From this we can deduce that in the funnel decomposition of P_i there are at most $0.2|P|/n^\alpha = O(|P|/n^\alpha)$ funnels (at the end of a maximal funnel there must be a short shortcut by the definition of a funnel).

Since in half of the calls to *Short-Shortcuts*(M_i), for $i = 1, \dots r$, the funnel decomposition of P_i has $O(|P|/n^\alpha)$ funnels, we get w.h.p. [24] that during *Update-Shortcuts*(M) we run *Long-Shortcuts*(M_j), for some $1 \leq j \leq r$, where P_j satisfies the above (i.e., has at most $|P|/n^\alpha$ funnels in its funnel decomposition).

Let M' be the matrix in which we accumulate long shortcuts in *Update-Shortcuts*(M). We prove in Lemma F.11 that if we run *Long-Shortcuts*(M_j) where P_j has $t = O(|P_j|/n^\alpha)$ funnels (and therefore $|P_j| = \Omega(tn^\alpha)$) in its decomposition then there is a monotone path P' in $G^{M'}$ from u to v that satisfies $|P'| \leq (1 - 1/\log n)\,|P_j|$ and $g^{G^{M'}}(P') \geq g^{M_j}(P_j)$, which proves the Lemma F.2.

[22]Note that an arc can be contained in at most two maximal funnels. Therefore it is important to specify which maximal funnel we pick.

[23]The same argument applies for descending paths but we will need to incorporate charge drops to the argument.

[24]The probability is $1 - \left(1 - \frac{\log n}{r}\right)^r \geq 1 - \frac{1}{n}$

To prove Lemma F.11, for every arc $e \in P_j$ we consider the furthest first-arc bounded subpath P_e of P that starts at e. Note that similarly to Lemma E.13, if we extend P_e with the next arc in P, we get a monotone path of length at least $|P_e|$. We prove in Lemma F.10 that the arc-bounded subpaths P_e for $e \in E$, form a laminar set. We then argue that there is a large subset $B \subseteq \{P_e \mid e \in P_j\}$ that satisfies

1. $|B| = \Omega(|P|/\log n)$
2. B has a stronger structure than laminarity: It is a union of chains $B = \cup_{k=1}^{q} B_k$, where a chain B_k is a set of subpaths such that for every two paths $P_1, P_2 \in B_k$ either $P_1 \subseteq P_2$ or $P_2 \subseteq P_1$.
3. There exists $0 \leq f^\star \leq \log n$ such that for every $P_e \in B$, it holds that the number of maximal funnels in P_e is at least $2^{f^\star}$ and less than $2^{f^\star+1}$.
4. Similarly to the bound $|P_j| = \Omega(tn^\alpha)$, i.e., the length of P_j is larger than the number of funnels in P_j by at least a factor of n^α, we have $|B_k| = \Omega\left(\frac{2^{f^\star} n^\alpha}{\log n}\right)$ for every $1 \leq k \leq q$. This means that the length of the longest path in B_k is larger by a factor of at least $n^\alpha/\log n$ than the number of funnels inside it.

Let $1 \leq k \leq q$. We finish the argument by saying that because of the chain structure of B_k and because we uniformly sample vertices in *Long-Shortcuts*, we will sample w.h.p. a vertex $v \in T_{f^\star}$ (see *Long-Shortcuts*) that is the first vertex of a path $P_e \in B_k$ that contains $\Omega(|B_k|)$ of the paths in B_k. In particular $|P_e| = \Omega(|B_k|)$. By Lemma E.13, after *Arc-Bounded-To-Monotone*(M, D), D will dominate the monotone path that corresponds to P_e, which is of length $|P_e| = \Omega(|B_k|)$. Because B is composed of disjoint chains, it follows that w.h.p. the total shortcutting we perform to P will be of size $\sum_{k=1}^{q} \Omega(|B_k|) = \Omega(|B|) = \Omega(|P|/\log n)$.
The proof of Lemma F.11 is based on the following structural definitions that formalize the paths P_e in the above explanation. These definitions allow us to measure how many applications of *Concatenate* and *Concatenate-Opposite* are needed in order to dominate a path P_e.

Definition F.4. *Let $P = e_1 \ldots e_k$ be a path in G^M. For every $1 \leq i \leq k$ we define*

- *$\bar{s}^P(i) \geq i$, the maximal index such that $e_i \ldots e_{\bar{s}^P(i)}$ is e_i-bounded.*
- *$\underline{s}^P(i) \leq i$, the smallest index such that $e_{\underline{s}^P(i)} \ldots e_i$ is e_i-bounded.*

When P is clear from the context, we abbreviate and write $\bar{s}(i), \underline{s}(i)$.

Definition F.5. *Let $P = e_1 \ldots e_k$ be a path in G^M and let $F_1, \ldots, F_t$ be the funnel decomposition of P. For every i we define*

- *$\bar{f}^P(i) = b - a + 1$, where a is maximal such that $e_i \in F_a$ and b is minimal such that $e_{\bar{s}(i)} \in F_b$.*
- *$\underline{f}^P(i) = a - b + 1$, where a is minimal such that $e_i \in F_a$ and b is maximal such that $e_{\underline{s}(i)} \in F_b$.*

When P is clear from context, we abbreviate and write $\bar{f}(i), \underline{f}(i)$.

Remark 1. *Some arcs on a path might belong to two funnels (arcs that end/start a funnel). This is the reason Definition F.5 needs to specify a concrete funnel that contains $e_i, e_{\bar{s}(i)}, e_{\underline{s}(i)}$.*

The following lemma states that for every arc e_i in a path $P = e_1 \ldots e_k$, if we extend the arc bounded path $e_i \ldots e_{\bar{s}(i)}$ by a single arc then we can extract from this path a monotone path, See Figure 20(a),(d) and (c),(f).

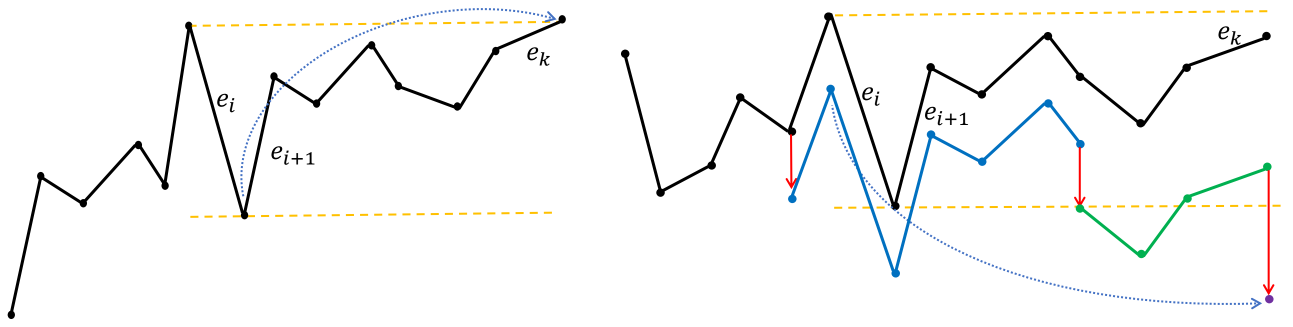


Figure 22: Two illustrations of Lemma F.7. The dotted blue arrows point from the beginning to the end of the monotone suffixes. On the left we have an ascending path $e_1 \dots e_k$, where $\bar{s}(i) = k$ and e_i has negative gain. On the right we have a descending path $e_1 \dots e_k$ (the original path is in black) with respect to a charge drop schedule (indicated by the down vertical red arrows), where $\bar{s}(i) = k$ and e_i has negative gain. The two symmetric cases in which e_i is of positive gain are not shown.

Lemma F.6. *Let $P = e_1 \dots e_k$ be a path in G^M. Then for every $1 < i < k$*

- *If $\bar{s}(i) < k$ then either $e_i \dots e_{\bar{s}(i)+1}$ is monotone or $e_{i+1} \dots v_{\bar{s}(i)+1}$ is monotone.*
- *If $\underline{s}(i) > 1$ then either $e_{\bar{s}(i)-1} \dots e_i$ is monotone or $e_{\bar{s}(i)-1} \dots e_{i-1}$ is monotone.*

Proof. TOPROVE 23 □

The following lemma is similar to Lemma F.6 and addresses the case in which a maximal arc bounded path in P reaches the last arc of P, and P is monotone with respect to a charge drop schedule, See Figure 22.

Lemma F.7. *Let $P = e_1 \dots e_k$ be a monotone path in G^M.*
If P is ascending, then for every $1 < i < k$

- *If $\bar{s}(i) = k$ then either $e_i \dots e_k$ is ascending or $e_{i+1} \dots e_k$ is ascending.*
- *If $\underline{s}(i) = 1$ then either $e_1 \dots e_i$ is ascending or $e_1 \dots e_{i-1}$ is ascending.*

If P is descending with respect to a charge drop schedule C, then for every $1 < i < k$

- *If $\bar{s}(i) = k$ then, with respect to an appropriate suffix of C, either $e_i \dots e_k$ is descending or $e_{i+1} \dots e_k$ is descending.*
- *If $\underline{s}(i) = 1$ then, with respect to an appropriate prefix of C, either $e_1 \dots e_i$ is descending or $e_1 \dots e_{i-1}$ is descending.*

Proof. TOPROVE 24 □

Lemma F.8. *Let $P = e_1 \dots e_k = v_1 \dots v_{k+1}$ be a negative arc bounded path in G^M. Let $F_1, \dots F_t$ be the funnel decomposition of P. The following holds w.h.p. after the main for-loop in Long-Shortcuts(M, D).*

- *Assume P is a $\overline{v_1}\underline{v_2}v_{k+1}$-path in G^M. If v_1 is sampled to T_j, where $2^j \geq t$, then D dominates P.*
- *Assume P is a $v_1\overline{v_k}\underline{v_{k+1}}$-path in G^M. If v_{k+1} is sampled to T_j, where $2^j \geq t$, then D dominates P.*

Proof. TOPROVE 25 □

The following is a corollary of Lemma E.13 and Lemmas F.6, F.7, F.8.

Corollary F.9. *Let $P = e_1 \dots e_k$ be a monotone path in G^M which is either descending with respect to a charge drop schedule C or ascending with respect to the zero schedule. Let $(u, v) = e_i \in P$ be a negative gain arc and let $\bar{P}_i, \underline{P}_i$ be the monotone*[25] *paths corresponding to $e_{\bar{s}(i)}, e_{\underline{s}(i)}$, respectively, given by Lemmas F.6 and F.7.*[26] *The following holds at the end of Long-Shortcuts(M).*

- *Assume $\bar{P}_i$ starts with e_i. If u is sampled into T_j and $2^j \geq \bar{f}(i)$, then D dominates $\bar{P}_i$.*
- *Assume $\bar{P}_i$ starts with e_{i+1}. If u is sampled into T_j and $2^j \geq \bar{f}(i)$, then D dominates $\bar{P}_i$.*
- *Assume $\underline{P}_i$ ends with e_i. If v is sampled into T_j and $2^j \geq \underline{f}(i)$, then D dominates $\underline{P}_i$.*
- *Assume $\underline{P}_i$ ends with e_{i-1}. If v is sampled into T_j and $2^j \geq \underline{f}(i)$, then D dominates $\underline{P}_i$.*

The domination is with respect to a sub-schedule of C.

Proof. TOPROVE 26 □

The following lemma proves that for every path $P = e_1 \dots e_k$, the set of paths $\{e_i \dots e_{\bar{s}(i)} \mid 1 \leq i \leq k\}$ is laminar and similarly $\{e_{\underline{s}(i) \dots e_i} \mid 1 \leq i \leq k\}$ is laminar.

Lemma F.10. *Let $P = e_1 \dots e_k$ be a path in G^M, then the sets of intervals $\{(i, \bar{s}(i)) \mid 1 \leq i \leq k\}$ and $\{(\underline{s}(i), i) \mid 1 \leq i \leq k\}$ are laminar.*

Proof. TOPROVE 27 □

The following lemma easily derives Lemma F.2. This lemma is our main theoretical contribution and the key to our result.

Lemma F.11. *Let $P = e_1 \dots e_k$ be a monotone simple path in G^M with respect to a charge drop schedule C, from s to t. Let $F_1, \dots, F_t$ be the funnel decomposition of P. Let $\bar{M}$ be the shortcuts table returned from Long-Shortcuts(M). If $t \leq k/n^\alpha$ and k is polynomial in $n = |V|$, then w.h.p. there is a monotone path P' in $G^{\bar{M}}$, with respect to a charge drop schedule C', from s to t in $G^{\bar{M}}$ that satisfies $g^{G^{\bar{M}}}(P') \geq g^{G^M}(P)$ and $|P'| \leq (1 - 1/\Omega(\log n)) \cdot |P|$.*

Proof. TOPROVE 28 □

We are ready to prove Lemma F.2.

Proof. TOPROVE 29 □

F.3 Running Time

Lemma F.12. *Procedure Compute-Funnels(M) terminates in expected $\tilde{\Theta}(n^{10/3})$ time.*

Proof. TOPROVE 30 □

Lemma F.13. *Procedure Compute-Shortcuts(G) terminates in expected $\tilde{\Theta}(n^{3.5})$ time.*

Proof. TOPROVE 31 □

[25]These paths may have a charge drop schedule assigned to them.

[26]Monotone paths given by Lemma F.6 start either at u or at v and end at $e_{\bar{s}(i)+1}$. Monotone paths given by Lemma F.7 start either at u or at v and end at e_k. Similarly monotone paths given by Lemma F.6 end either at u or at v and start at $e_{\underline{s}(i)-1}$. Monotone paths given by Lemma F.7 end either at u or at v and start at e_1.

G Relating M and D to G

In Theorem F.1 we have seen that every monotone simple path in G is dominated w.h.p. by the final shortcuts table M returned by *Compute-Shortcuts*. Moreover, by Invariant 1 we know that every value in D is realizable by a traversable path in G^M.

The following lemma gives the relation between G^M and G. The lemma states that any traversable path in G^M can be "unwrapped" to a traversable path in G that has "better" α (maximum final charge) values.

Lemma G.1. *Let M be the shortcut table return by Compute-Shortcuts. Let $P = v_1 \dots v_k$ be a traversable path in G^M and let C be a charge drop schedule for P. There exists a traversable path $P' = P^{v_1v_2} \mid P^{v_2v_3} \mid \dots \mid P^{v_{k-1}v_k}$ in G and a charge drop schedule $C' = C^{v_1v_2} \mid C^{v_2v_3} \mid \dots \mid C^{v_{k-1}v_k}$ such that*

(a) $P^{v_iv_{i+1}}$ is a monotone path from v_i to v_{i+1} in G with respect to the charge drop schedule $C^{v_iv_{i+1}}$, for every $1 \leq i < k$. In particular, if P is of length 1 then P' is monotone with respect to C'.

(b) $g_{v_i}^{P,C} = g_{v_i}^{P',C'}$, for every $1 \leq i \leq k$.

(c) $\alpha_b^G(P') \geq \alpha_b^{G^M}(P)$ for every $b \in [0, B]$.

Proof. TOPROVE 32 □

We get as a corollary the following structural lemma about paths realizing the values in D.

Corollary G.2. *Let M be a shortcuts table and let D be a data structure that maintains Invariant 1 with respect to G^M. The following holds for every $x, y, z \in V$.*

1. *Assume $D[xy][z] \neq -\infty$. Then there exists a traversable path $P = P^{xy} \mid P^{yz}$ in G and a charge drop schedule $C = C^{xy} \mid C^{yz}$ such that*[27]

   (a) $g^C(P) = D[xy][z]$,

   (b) P^{xy} is monotone with respect to C^{xy} and $M[x][y] = g^{C^{xy}}(P^{xy})$,

   (c) The gains of the first and last vertices of P^{xy} (i.e. x and y) bound the gains of all other vertices in P. All gains are with respect to C.

2. *Assume $D[x][yz] \neq -\infty$. Then there exists a traversable path $P = P^{xy} \mid P^{yz}$ in G and a charge drop schedule $C = C^{xy} \mid C^{yz}$ such that*

   (a) $g^C(P) = D[x][yz]$,

   (b) P^{yz} is monotone with respect to C^{yz} and $M[y][z] = g^{C^{yz}}(P^{yz})$,

   (c) The gains of the first and last vertices of P^{yz} (i.e. y and z) bound the gains of all other vertices in P. All gains are with respect to C.

3. *Assume $D[x][y] \neq -\infty$. Then there exists a traversable path P in G and a charge drop schedule C such that*

   (a) $g^C(P) = D[x][y]$,

   (b) P is monotone with respect to C.

Proof. TOPROVE 33 □

[27]The paths P^{xy} and P^{yz} are paths from x to y and from y to z, respectively. We use the same convention also for claims 2 and 3.

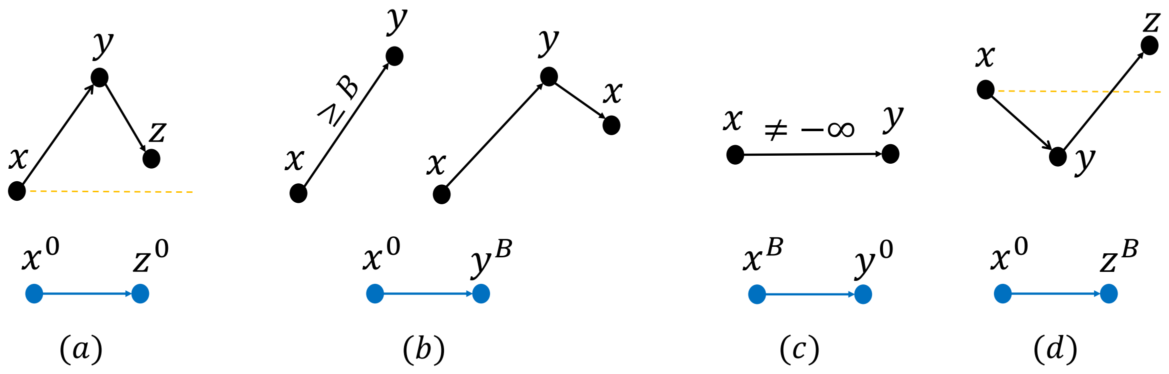


Figure 23: The 4 types of edges we include in H.

H Stage II - Computing the α values

Let M be the shortcuts table we receive from Stage I and let $D = \mathit{Compute\text{-}Funnels}(M)$.

In this appendix, using M and the data structure D, we show how to compute $\alpha_B(s,t)$ for every $s,t \in V$. Recall that $\alpha_B(s,t)$ is the maximum final charge at t when the car starts at s with a full battery. The algorithms proceeds in two steps.

In the first step we build a graph $H = (V^0 \cup V^B, E(H))$, where $V^b = \{v^b \mid b \in \{0,B\}\}$ represents that we are at v with at least b charge. An arc $u^{b_1}v^{b_2} \in E(H)$ represents that $\alpha_{b_1}(u,v) \geq b_2$.[28] We create the arcs $E(H) \subseteq \{u^{b_1}v^{b_2} \mid \alpha_{b_1}(u,v) \geq b_2\}$ by observing simple properties of the values in D. Finally we compute the transitive closure $H^\star$ of H. We claim in Theorem H.12 that w.h.p., for every $s,t \in V$, $\alpha_B(s,t) = B$ if and only if $s^Bt^B \in E(H^\star)$.

The second (and final) step is based on combining the following observations. Let $s,t \in V$ and let $P = v_1(= s)\dots v_k(= t)$ be an optimal path from s to t (i.e., $\alpha_B(s,t) = \alpha_B(P)$). If $g_{v_i} < 0$ for every $i \leq k$ then we can assume that P is simple (otherwise it contains a positive gain cycle and we can repeat this cycle to improve final charge) and we show in Lemma H.15 that $\alpha_B(s,t)$ is realized by a funnel in G^M. Otherwise, some vertices in P are visited with full charge. Using $H^\star$ from the first step (Theorem H.12), we can find the last vertex $y \in P$ that is reached with full charge and compute $\alpha_B(y,t)$. We claim that the suffix P^{yt} of P from y to t is simple, so (by Lemma H.15) $\alpha_B(y,t)$ is realized by a funnel in G^M. Thus, the second step amounts to finding pairs (y,t) such that $s^By^B \in H^\star$ and $\alpha_B(y,t)$ can be realized by a funnel. We use the best such pairs in order to compute $\alpha_B(s,t)$ for every $s,t \in V$.

The rest of this section is organized as follows. In Appendix H.1 we build the transitive closure graph $H^\star$ and prove basic properties of $H^\star$. In Appendix H.2 we prove that $H^\star$ indeed finds all $s,t \in V$ such that $\alpha_B(s,t) = B$. Finally, in Appendix H.3 we complete the computation of $\alpha_B(\cdot,\cdot)$ and prove its correctness as described above.

H.1 The transitive closure graph

After performing $\mathit{Compute\text{-}Shortcuts}(G)$ we received a table M of shortcuts and computed the data structure $D = \mathit{Compute\text{-}Funnels}(M)$. Using M and D, we construct the graph H. In the following sections we define the arcs of H, see Figure 23. After building H, we compute its transitive closure graph $H^\star$.

[28]Note that the other direction does not necessarily hold: It is possible that $\alpha_{b_1}(u,v) \geq b_2$ but $u^{b_1}v^{b_2} \notin E(H)$.

H.1.1 0-0 arcs

For every $x, y, z \in V$, add an arc x^0z^0 to $E(H)$ if $M[x][y] + M[y][z] \geq 0$ and $M[x][y] \geq 0$. See Figure 23(a).

Lemma H.1. *Let $x^0z^0 \in E(H)$, then $\alpha_0(x, z) \geq 0$.*

Proof. TOPROVE 34 □

H.1.2 0-B arcs

For every $x, y \in V$, add an arc x^0y^B to $E(H)$ if either $M[x][y] \geq B$, or $M[x][y] + M[y][x] > 0$ and $M[x][y] > 0$. See Figure 23(b).

Lemma H.2. *Let $x^0y^B \in E(H)$, then $\alpha_0(x, y) = B$.*

Proof. TOPROVE 35 □

H.1.3 B-0 arcs

For every $x, y \in V$, add an arc x^By^0 to $E(H)$ if $M[x][y] \neq -\infty$. See Figure 23(c).

Lemma H.3. *Let $x^By^0 \in E(H)$, then $\alpha_B(x, y) \geq 0$.*

Proof. TOPROVE 36 □

H.1.4 B-B arcs

For every $x, y, z \in V$, add an arc x^Bz^B to $E(H)$ if $M[x][y] + M[y][z] \geq 0$ and $M[y][z] \geq 0$. See Figure 23(d).

Lemma H.4. *Let $x^Bz^B \in E(H)$, then $\alpha_B(x, z) = B$.*

Proof. TOPROVE 37 □

The following theorem is an immediate consequence of Lemmas H.1, H.2, H.3 and H.4.

Theorem H.5. *Let $x^{b_1}y^{b_2} \in E(H^\star)$, then $\alpha_{b_1}(x, y) \geq b_2$.*

H.2 Transitive closure graph - correctness

In this appendix we show that for every $s, t \in V$ it holds that $\alpha_B(s, t) = B$ if and only if $s^Bt^B \in E(H^\star)$, see Theorem H.11. We begin by addressing entry-exit pairs on positive gain cycles, see Definition C.4.

Lemma H.6. *Let C be a positive gain simple cycle in G. There exists an entry-exit pair (x, y) in C such that w.h.p. $x^0y^B \in E(H)$.*

Proof. TOPROVE 38 □

Lemma H.7. *Let P be a strongly traversable simple path in G from x to y, then w.h.p. $x^0y^0 \in E(H^\star)$.*

Proof. TOPROVE 39 □

Lemma H.8. *Let P be a simple path from x to y such that $\alpha_B(P) = B$, then w.h.p. $x^By^B \in E(H^\star)$.*

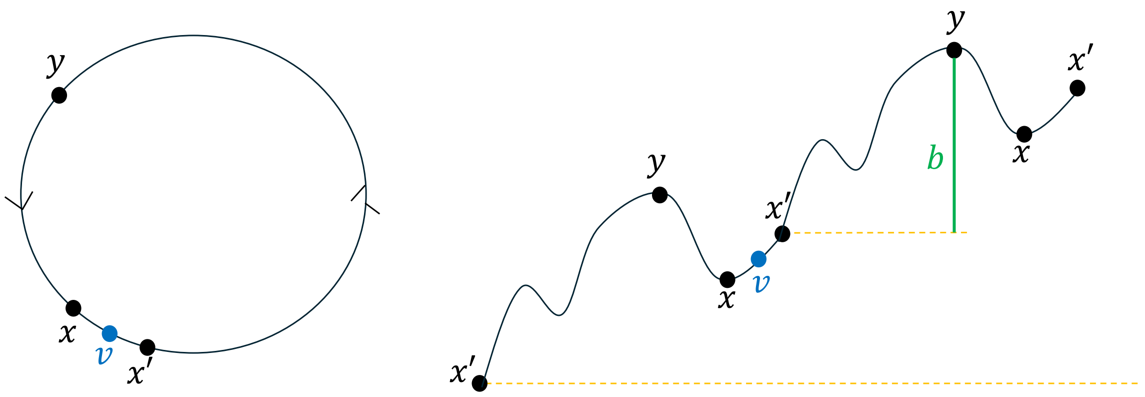


Figure 24: Illustration of Lemma H.6. Note that y is of maximum gain in the path from x' to itself (through the cycle) and that x is of minimum gain on the subpath from y to x'. As shown in the proof of Lemma H.6, the path from y to x is descending and the path from x to y is ascending.

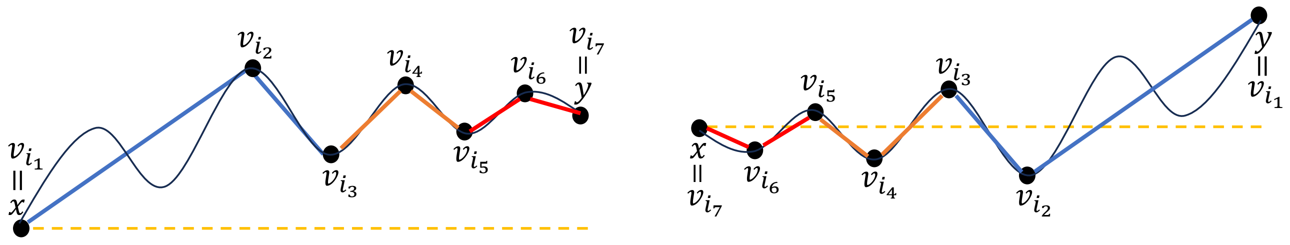


Figure 25: Right: The path decomposition in Lemma H.7. Note that we can pair ascending and descending paths and have overall nonnegative gain. Left: The path decomposition in Lemma H.8. We pair descending paths with ascending paths such that the descending path comes first.

Proof. TOPROVE 40 □

The following theorem states that we have indeed found all entry-exit pairs.

Lemma H.9. *Let (x, y) be an entry-exit pair in a positive gain cycle C, then w.h.p. x^0y^B is an arc in $H^\star$.*

Proof. TOPROVE 41 □

We are now ready to prove the main claim.

Theorem H.10. *Let $s, t \in V$. If $\alpha_0(s, t) = B$ then w.h.p. $s^0t^B \in E(H^\star)$.*

Proof. TOPROVE 42 □

Theorem H.11. *Let $s, t \in V$. If $\alpha_B(s, t) = B$ then w.h.p. $s^Bt^B \in E(H^\star)$.*

Proof. TOPROVE 43 □

By combining Theorem H.11 with Theorem H.5 we get the following theorem.

Theorem H.12. *For every $s, t \in V$, w.h.p., $\alpha_B(s, t) = B$ if and only if $s^Bt^B \in E(H^\star)$.*

H.3 Computing the $\alpha_B(\cdot, \cdot)$ values

The algorithm $MFC(M)$ for deriving of the $\alpha_B(\cdot, \cdot)$ values is given in Figure 26. The algorithm computes a table $\alpha_B[\cdot][\cdot]$ and we prove in Theorem H.16 that $\alpha_B[s][t] = \alpha_B(s, t)$, for every $s, t \in V$. Algorithm $MFC(M)$ starts by computing $H^\star$ as explained in Appendix H.1.

The algorithm is based on the following idea. For $s, t \in V$, let $P = v_1(= s) \dots v_k(= t)$ be an optimal path from s to t (i.e., $\alpha_B(P) = \alpha_B(s, t)$) that follows the structure of Lemma C.5 and let $C_1, \dots C_\ell$ and $(x_1, y_1), \dots (x_\ell, y_\ell)$ as in Lemma C.5. Let $1 \leq i \leq k$ be the maximum index that satisfies $\alpha_B(v_1 \dots v_i) = B$. By Theorem H.12, w.h.p. $s^Bv_i^B \in E(H^\star)$. By Lemma C.5 it holds that $\alpha_B(v_1, y_\ell) = B$ and therefore $v_i \in P^{y_\ell v_k}$, where $P^{y_\ell v_k}$ is the (simple) subpath of P from y_ℓ to t. Hence $v_i \dots v_k$ is a simple path.

We prove in Lemma H.15 that there exists $x \in V$ that satisfies $M[v_i][x] \in [-B, 0]$ and $B + D[v_ix][v_k] = \alpha_B(v_i, v_k)(= \alpha_B(v_1, v_k))$, see Figure 27 where $y = v_i, x = v_{i_2}, v_k = t$.

Based on the above, the algorithm proceeds as follows. For every $y, t \in V$, we upper bound the largest final charge we can get if we use a simple path P that starts at y with full charge and ends at t such that y has the maximum gain in P. We store these values in a table A_B whose computation is done by assigning $A_B[y][t] \leftarrow \max\{B + D[yx][t] \mid x \in V,\ M[y][x] \leq 0\}$, for every $y, t \in V$. Finally, the computation of $\alpha_B[s][t]$ is done by assigning $\alpha_B[s][t] \leftarrow \max\{A_B[y][t] \mid s^By^B \in E(H^\star)\}$.

The following lemma states that the $A_B[\cdot][\cdot]$ values $MFC(M)$ computes lower bound the actual $\alpha_B(\cdot, \cdot)$ values.

Lemma H.13. *Let M be the shortcut table returned by Compute-Shortcuts. Let $D =$ Compute-Funnels(M) and let $\alpha_B[\cdot][\cdot]$ be the result of $MFC(M)$. Then $\alpha_B(y, t) \geq B + D[yx][t]$ for every $y, x, t \in V$ that satisfy $-B \leq M[y][x] \leq 0$. In particular, $A_B[y][t] \leq \alpha_B(y, t)$ for every $y, t \in V$.*

Proof. TOPROVE 44 □

As a corollary, we get that the $\alpha_B[\cdot][\cdot]$ values that $MFC(M)$ computes, lower bound the actual $\alpha_B(\cdot, \cdot)$ values.

Corollary H.14. *For every $s, t \in V$ it holds that $\alpha_B[s][t] \leq \alpha_B(s, t)$.*

```
$MFC(M)$:
    $H \leftarrow Build\_H(M)$                                  // As explained in Appendix H.1
    $H^\star \leftarrow Transitive\_closure(H)$
    $D \leftarrow Compute\text{-}Funnels(M)$
    $A_B \leftarrow matrix(n, n, -\infty)$  // $\alpha_B(\cdot,\cdot)$ of simple bounded paths starting with $B$ charge
    for $y, t \in V$ do
        for $x \in V$ do
            if $M[y][x] \leq 0$ : // $\bar{y}xt$ paths
                $A_B[y][t] \leftarrow \max\{A_B[y][t], B + D[yx][t]\}$
    $\alpha_B \leftarrow matrix(n, n, -\infty)$
    for $s, t \in V$ do
        if $s^B t^B \in E(H^\star)$ :
            $\alpha_B[s][t] \leftarrow B$
        for $y \in V$ do
            if $s^B y^B \in E(H^\star)$ :
                $\alpha_B[s][t] \leftarrow \max\{\alpha_B[s][t], A_B[y][t]\}$
    return $\alpha$
```

Figure 26: Computing the *maximum final charges* $\alpha_B(s,t)$ for every $s, t \in V$.

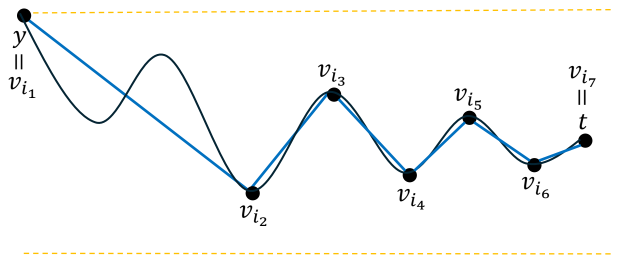


Figure 27: The path decomposition in Lemma H.15. The blue arcs correspond to arcs in G^M of **the same** gain as the subpaths.

Proof. TOPROVE 45 □

Lemma H.15. *Let $y, t \in V$. If there is a simple traversable path P from y to t such that $\alpha_B(P) = \alpha_B(y,t)$ and $g_v < g_y = 0$ for every $(y \neq)v \in P$, then $\alpha_B(y,t) = A_B[y][t]$.*

Proof. TOPROVE 46 □

Theorem H.16. *Algorithm $MFC(M)$ computes $\alpha_B(s,t)$ for every $s, t \in V$.*

Proof. TOPROVE 47 □

The following theorem summarise the main result of this paper.

Theorem H.17. *Let $G = (V, A, g)$ be a road network that may contain positive gain cycles and let $B \in \mathbb{R}^+$. There is a randomized algorithm that in expected $\tilde{O}(n^{3.5})$ time computes a table $\alpha_B[\cdot][\cdot]$ such w.h.p. $\alpha_B[s][t] = \alpha_B(s,t)$ for every $s, t \in V$.*