Two-phase algorithms for the parametric shortest path problem

Sourav Chakraborty[1], Eldar Fischer[1], Oded Lachish[2], and Raphael Yuster[3]

[1] Department of Computer Science, Technion, Haifa 32000, Israel
{eldar,sourav}@cs.technion.ac.il
Research supported in part by an ERC-2007-StG grant number 202405-2
[2] Centre for Discrete Mathematics and its Applications, University of Warwick , Coventry, UK
oded@dcs.warwick.ac.uk
[3] Department of Mathematics, University of Haifa, Haifa 31905, Israel
raphy@math.haifa.ac.il

Abstract. A *parametric weighted graph* is a graph whose edges are labeled with continuous real functions of a single common variable. For any instantiation of the variable, one obtains a standard edge-weighted graph. Parametric weighted graph problems are generalizations of weighted graph problems, and arise in various natural scenarios. Parametric weighted graph algorithms consist of two phases. A *preprocessing phase* whose input is a parametric weighted graph, and whose output is a data structure, the advice, that is later used by the *instantiation phase*, where a specific value for the variable is given. The instantiation phase outputs the solution to the (standard) weighted graph problem that arises from the instantiation. The goal is to have the running time of the instantiation phase supersede the running time of any algorithm that solves the weighted graph problem from scratch, by taking advantage of the advice.

In this paper we construct several parametric algorithms for the shortest path problem. For the case of linear function weights we present an algorithm for the single source shortest path problem. Its preprocessing phase runs in $\tilde{O}(V^4)$ time, while its instantiation phase runs in only $O(E + V \log V)$ time. The fastest standard algorithm for single source shortest path runs in $O(VE)$ time. For the case of weight functions defined by degree d polynomials, we present an algorithm with sub-exponential preprocessing time $O(V^{(1+\log f(d))\log V})$ and instantiation time only $\tilde{O}(V)$. In fact, for any pair of vertices u, v, the instantiation phase computes the distance from u to v in only $O(\log^2 V)$ time. Finally, for linear function weights, we present a randomized algorithm whose preprocessing time is $\tilde{O}(V^{3.5})$ and so that for any pair of vertices u, v and any instantiation variable, the instantiation phase computes, in $O(1)$ time, a length of a path from u to v that is at most (additively) ϵ larger than the length of a shortest path. In particular, an all-pairs shortest path solution, up to an additive constant error, can be computed in $O(V^2)$ time.

1 Introduction

A *function-weighted graph* is a graph whose edges are labeled with continuous real functions. When all the functions are univariate (and all have the same variable), the graph is called a *parametric weighted graph*. In other words, the graph is $G = (V, E, W)$ where $W : E \to \mathcal{F}$ and $\mathcal{F}$ is the space of all real continuous functions with the variable x. If G is a parametric weighted graph, and $r \in \mathbb{R}$ is any real number, then $G(r)$ is the standard weighted graph where the weight of an edge e is defined to be $(W(e))(r)$. We say that $G(r)$ is an *instantiation* of G, since the variable x in each function is instantiated by the value r. Parametric weighted graphs, are, therefore, a generic instance of infinitely many instances of weighted graphs.

How can we benefit from modeling problems by a parametric weighted graph? If we can take advantage of the generic instance G, and use it to precompute some general generic information $I(G)$, it is plausible that for any given instantiation $G(r)$, we will be able to use the precomputed information $I(G)$ in order to speed up the time to solve the given problem on $G(r)$, faster than just solving the problem on $G(r)$ from scratch. Let us make this notion more precise.

A *parametric weighted graph algorithm* (or, for brevity, *parametric algorithm*) consists of two phases. A *preprocessing phase* whose input is a parametric weighted graph G, and whose output is a data structure (the advice) that is later used by the *instantiation phase*, where a specific value r for the variable is given. The instantiation phase outputs the solution to the (standard) weighted graph problem on the weighted graph $G(r)$. Naturally, the goal is to have the running time of the instantiation phase significantly smaller than the running time of any algorithm that solves the weighted graph problem from scratch, by taking advantage of the advice constructed in the preprocessing phase. Parametric algorithms are therefore evaluated by a pair of running times, the *preprocessing time* and the *instantiation time*.

In this paper we show that parametric algorithms are beneficial for one of the most natural combinatorial optimization problems: the *shortest path* problem in directed graphs. Recall that given a directed real-weighted graph G, and two vertices u, v of G, the distance from u to v, denoted by $\delta(u, v)$, is the length of a shortest path from u to v. The *single pair* shortest path problem seeks to compute $\delta(u, v)$ and construct a shortest path from u to v. Likewise, the *single source* shortest path problem seeks to compute the distances and shortest paths from a given vertex to all other vertices, and the *all pairs* version seeks to compute distances and shortest paths between all ordered pairs of vertices. In some of our algorithms we forgo the calculation of the path itself to achieve a shorter instantiation time. In all those cases the algorithms can be easily modified to also output a shortest path, in which case their instantiation time is the sum of the time it takes to calculate the distance and a time linear in the size of the path to be output.

Our first algorithm is a parametric algorithm for single source shortest path, in the case where the weights are *linear* functions. That is, each edge e is labeled with a function $a_e x + b_e$ where a_e and b_e are reals. Such linear parametrization has practical importance. Indeed, in many problems the cost of an edge is composed from some constant term plus a term which is a factor of some commodity, whose cost varies (e.g. bank commissions, taxi fares, vehicle maintenance costs, and so on). Our parametric algorithm has preprocessing time $\tilde{O}(n^4)$ and instantiation time $O(m + n \log n)$ (throughout this paper n and m denote the number of vertices and edges of a graph, respectively). We note that the fastest algorithm for the single source shortest path in real weighted directed graphs requires $O(nm)$ time; the Bellman-Ford algorithm [2]. The idea of our preprocessing stage is to precompute some other linear functions, on the *vertices*, so that for every instantiation r, one can quickly determine whether $G(r)$ has a negative cycle and otherwise use these functions to quickly produce a reweighing of the graph so as to obtain only nonnegative weights similar to the weights obtained by Johnson's algorithm [11]. In other words, we *avoid* the need to run the Bellman-Ford algorithm in the instantiation phase. The $\tilde{O}(n^4)$ time in the preprocessing phase comes from the use of an interior point algorithm that we need in order to compute the linear vertex functions.

Theorem 1. *There exists a parametric algorithm for single source shortest path in graphs weighted by linear functions, whose preprocessing time is $\tilde{O}(n^4)$ and whose instantiation time is $O(m + n \log n)$.*

Our next algorithm applies to a more general setting where the weights are polynomials of degree at most d. Furthermore, in this case our goal is to have the instantiation phase answer distance queries between any two vertices in *sublinear* time. Notice first that if we allow exponential preprocessing time, this goal can be easily achieved. This is not hard to see by the fact that the overall possible number of shortest paths (when x varies over the reals) is $O(n!)$, or from Fredman's decision tree for shortest paths whose height is $O(n^{2.5})$ [8]. But can we settle for *sub-exponential* preprocessing time and still be able to have sublinear instantiation time? Our next result achieves this goal.

Theorem 2. *There exists a parametric algorithm for the single pair shortest path problem in graphs weighted by degree d polynomials, whose preprocessing time is $O(n^{(O(1)+\log f(d)) \log n})$ and instantiation time $O(\log^2 n)$, where $f(d)$ is the time required to compute the intersection points of two degree d polynomials. The size of the advice that the preprocessing algorithm produces is $O(n^{(O(1)+\log d) \log n})$.*

The practical and theoretical importance of shortest path problems lead several researchers to consider fast algorithms that settle for an approximate shortest path. For the general case (of real weighted digraphs) most of the algorithms guarantee an α-*stretch* factor. Namely, they compute a path whose length is at most $\alpha\delta(u,v)$. We mention here the $(1+\epsilon)$-stretch algorithm of Zwick for the all-pairs shortest path problem, that runs in $\tilde{O}(n^{\omega})$ time when the weights are non-negative reals [16]. Here $\omega < 2.376$ is the matrix multiplication exponent [5].

Here we consider probabilistic additive-approximation algorithms, or *surplus* algorithms, that work for linear weights which may have positive and negative values (as long as there is no negative weight cycle). We say that a shortest path algorithm has an ϵ-surplus if it computes paths whose lengths are at most $\delta(u,v)+\epsilon$. We are unaware of any truly subcubic algorithm that guarantees an ϵ-surplus approximation, and which outperforms the fastest general all-pairs shortest path algorithm [4].

In the linear-parametric setting, it is easy to obtain ϵ-surplus parametric algorithms whose preprocessing time is $O(n^4)$ time, and whose instantiation time, for any ordered pair of queried vertices u,v is constant. It is assumed instantiations are taken from some interval I whose length is independent of n. Indeed, we can partition I into $O(n)$ subintervals $I_1, I_2, \ldots$ of size $O(1/n)$ each, and solve, in cubic time (say, using [7]), the exact all-pairs solution for any instantiation r that is an endpoint of two consecutive intervals. Then, given any $r \in I_j = (a_j, b_j)$, we simply look at the solution for b_j and notice that we are (additively) off from the right answer only by $O(1)$. Standard scaling arguments can make the surplus smaller than ϵ. But do we really need to spend $O(n^4)$ time for preprocessing? In other words, can we invest (significantly) less than $O(n^4)$ time and still be able to answer instantiated distance queries in $O(1)$ time? The following result gives a positive answer to this question.

Theorem 3. *Let $\epsilon > 0$, let $[\alpha,\beta]$ be any fixed interval and let γ be a fixed constant. Suppose G is a linear-parametric graph that has no negative weight cycles in the interval $[\alpha,\beta]$, and for which every edge weight $a_e + xb_e$ satisfies $|a_e| \le \gamma$. There is a parametric randomized algorithm for the ϵ-surplus shortest path problem, whose preprocessing time is $\tilde{O}(n^{3.5})$ and whose instantiation time is $O(1)$ for a single pair, and hence $O(n^2)$ for all pairs.*

We note that this algorithm works in the restricted addition-comparison model. We also note that given an ordered pair u,v and $r \in [\alpha,\beta]$, the algorithm outputs, in $O(1)$ time, a weight of an actual path from u to v in $G(r)$, and points to a linked list representing that path. Naturally, if one wants to output the vertices of this path then the time for this is linear in the length of the path.

The rest of this paper is organized as follows. The next subsection shortly surveys related research on parametric shortest path problems. In the three sections following it we prove Theorems 1,2 and 3. Section 5 contains some concluding remarks and open problems.

1.1 Related research

Karp and Orlin [13], and, later, Young, Tarjan, and Orlin [15] considered a special case of the linear-parametric shortest path problem. In their case, each edge weight e is either some fixed constant b_e or is of the form $b_e - x$. It is not too difficult to prove that for any given vertex v, when x varies from $-\infty$ to the largest x_0 for which $G(x_0)$ has no negative weight cycle (possibly $x_0 = \infty$), then there are at most $O(n^2)$ distinct shortest path trees from v to all other vertices. Namely, for each $r \in [-\infty, x_0]$ one of the trees in this family is a solution for single-source shortest path in $G(r)$. The results in [13, 15] cleverly and compactly compute all these trees, and the latter does it in $O(nm + n^2 \log n)$ time.

However, for general linear functions, Carstensen [3] showed that there are constructions for which the number of shortest path changes while x varies over the reals is $n^{\Omega(\log n)}$. In fact, in her example each linear function is of the form $a_e + xb_e$ and both a_e and b_e are positive, and x varies in $[0, \infty]$. Carstensen also proved that this is tight. In other words, for any linear-parametric graph the number of changes in the shortest paths is $n^{O(\log n)}$. A simpler proof was obtained by Nikolova et al. [14], that also supply an $n^{O(\log n)}$ time algorithm to compute the path breakpoints. Their method, however, does not apply to the case where the functions are not linear, such as in the case of degree d polynomials.

Gusfield [10] studied algorithms for function-weighted graphs but in the context of program module distribution.

2 Proof of Theorem 1

The proof of Theorem 1 follows from the following two lemmas.

Lemma 1. *Given a linear-weighted graph $G = (V, E, W)$, there exist $\alpha, \beta \in \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$ such that $G(r)$ has no negative cycles if and only if $\alpha \le r \le \beta$. Moreover α and β can be found in $\tilde{O}(n^4)$ time.*

Lemma 2. *Let $G = (V, E, W)$ be a linear-weighted graph. Also let $\alpha, \beta \in \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$ be such that at least one of them is finite and for all $\alpha \ge r \ge \beta$ the graph $G(r)$ has no negative cycle. Then for every vertex $v \in V$ there exists a linear function $g_v^{[\alpha,\beta]}$ such that if the new weight function W' is given by*

$$W'((u, v)) = W((u, v)) + g_u^{[\alpha,\beta]} - g_v^{[\alpha,\beta]}$$

then the new linear-weighted graph $G' = (V, E, W')$ has the property that for any real $\alpha \le r \le \beta$ all the edges in $G'(r)$ are non-negative. Moreover the functions $g_v^{[\alpha,\beta]}$ for all $v \in V$ can be found in $O(mn)$ time.

So given a linear-weighted graph G, we first use Lemma 1 to compute α and β. If at least one of α and β is finite then using Lemma 2 we compute the n linear functions $g_v^{[\alpha,\beta]}$, one for each $v \in V$. If $\alpha = -\infty$ and $\beta = +\infty$, then using Lemma 2 we compute the $2n$ linear functions $g_v^{[\alpha,0]}$ and $g_v^{[0,\beta]}$. These linear functions will be the advice that the preprocessing algorithm produces. The above lemmas guarantee us that the advice can be computed in time $\tilde{O}(n^4)$, that is the preprocessing time is $\tilde{O}(n^4)$.

Now when computing the single source shortest path problem from vertex v for the graph $G(r)$ our algorithm proceeds as follows:

1. If $r < \alpha$ or $r > \beta$ output "$-\infty$" as there exists a negative cycle (such instances are considered invalid).
2. If $\alpha \le r \le \beta$ and at least one of α or β is finite then compute $g_u(r)$ for all $u \in V$. Use these to re-weight the edges in the graph as in Johnson's algorithm [11]. If $\alpha = -\infty$ and $\beta = +\infty$ then if $r \le 0$ compute $g_u^{[\alpha,0]}(r)$ for all $u \in V$ and if $r \ge 0$ compute $g_u^{[0,\beta]}(r)$ for all $u \in V$. Notice that after the reweighing we have an instance of $G'(r)$.
3. Use Dijkstra's algorithm [6] to solve the single source shortest path problem in $G'(r)$. Dijkstra's algorithm applies since $G'(r)$ has no negative weight edges. The shortest paths tree returned by Dijkstra's algorithms applied to $G'(r)$ is also the shortest paths tree in $G(r)$. As in Johnson's algorithm, we use the results $d'(v, u)$ of $G'(r)$ to deduce $d(v, u)$ in $G(r)$ since, by Lemma 2 $d(v, u) = d'(v, u) - g_v(r) + g_u(r)$.

The running time of the instantiation phase is dominated by the running time of Dijkstra's algorithm which is $O(m + n \log n)$ [9].

2.1 Proof of Lemma 1

Since the weight on the edges of the graph G are linear functions, we have that the weight of any directed cycle in the graph is also a linear function. Let $C_1, C_2, \ldots, C_T$ be the set of all directed cycles in the graph. The linear weight function of a cycle C_i will be denoted by $\mathrm{wt}(C_i)$. If $\mathrm{wt}(C_i)$ is not the constant function, then let γ_i be the real number for which the linear equation $\mathrm{wt}(C_i)$ evaluates to 0.
Let α and β be defined as follows:

$$\alpha = \max_i \{\gamma_i \mid \ \mathrm{wt}(C_i) \text{ has a positive slope}\}.$$

$$\beta = \min_i \{\gamma_i \mid \ \mathrm{wt}(C_i) \text{ has a negative slope}\}.$$

Note that if $\mathrm{wt}(C_i)$ has a positive slope then

$$\gamma_i = \min_x \{\mathrm{wt}(C_i)(x) \geq 0\}.$$

Thus for all $x \geq \gamma_i$ the value of $\mathrm{wt}(C_i)$ evaluated at x is non-negative. So by definition for all $x \geq \alpha$ the value of the $\mathrm{wt}(C_i)$ is non-negative if the slope of $\mathrm{wt}(C_i)$ is positive, and for any $x < \alpha$ there exists a cycle C_i such that $\mathrm{wt}(C_i)$ has positive slope and $\mathrm{wt}(C_i)(x)$ is negative. Similarly, for all $x \leq \beta$ the value of the $\mathrm{wt}(C_i)$ is non-negative if the slope of $\mathrm{wt}(C_i)$ is negative and for any $x > \beta$ there exists a cycle C_i such that $\mathrm{wt}(C_i)$ has negative slope and $\mathrm{wt}(C_i)(x)$ is negative.

This proves the existence of α and β. There are, however, two bad cases that we wish to exclude. Notice that if $\alpha > \beta$ this means that for any evaluation at x, the resulting graph has a negative weight cycle. The same holds if there is some cycle for which $\mathrm{wt}(C_i)$ is constant and negative. Let us now show how α and β can be efficiently computed whenever these bad cases do not hold. Indeed, α is the solution to the following Linear Program (LP), which has a feasible solution if and only if the bad cases do not hold.

Minimize x under the constraints
$\forall i$**,** $\mathrm{wt}(C_i)(x) \geq 0$.

This is an LP on one variable, but the number of constraints can be exponential. Luckily, however, the set of constraints can be presented in a compact form: our function-weighted graph. In other words, for any given x, one can use the Bellman-Ford algorithm to produce a constraint that is violated (if there is one), that is produce a cycle that has negative weight. Thus, our LP has an efficient separation oracle. As this LP has only one variable, and its size is only $O(n^2)$ (in this context the size is defined by the length of the parameters of the separation oracle, in our case the linearly weighted graph), it can be solved in $\tilde{O}(n^4)$ time [12]. Notice that β can be computed using the analogous maximization linear program. In case one (and hence both) of the linear programs has no feasible solution we just set $\alpha = \beta = \infty$.

2.2 Proof of Lemma 2

Let α and β be the two numbers such that for all $\alpha \leq r \leq \beta$ the graph $G(r)$ has no negative cycles and at least one of α and β is finite.

First let us consider the case when both α and β are finite. Recall that, given any number r, Johnson's algorithm associates a weight function $h^r : V \to \mathbb{R}$ such that, for any edge $(u, v) \in E$,

$$W_{(u,v)}(r) + h^r(u) - h^r(v) \geq 0.$$

(Johnson's algorithm computes this weight function by running the Bellman-Ford algorithm over $G(r)$). Define the weight function $g_v^{[\alpha,\beta]}$ as

$$g_v^{[\alpha,\beta]}(x) = \left(\frac{h^\beta(v) - h^\alpha(v)}{\beta - \alpha}\right) x + h^\alpha(v) - \left(\frac{h^\beta(v) - h^\alpha(v)}{\beta - \alpha}\right) \alpha \ .$$

This is actually the equation of the line joining $(\alpha, h^\alpha(v))$ and $(\beta, h^\beta(v))$ in $\mathbb{R}^2$.
Now we need to prove that for every $\alpha \leq r \leq \beta$ and for every $(u,v) \in V$,

$$W_{(u,v)}(r) + g_u^{[\alpha,\beta]}(r) - g_v^{[\alpha,\beta]}(r) \geq 0 \ .$$

Since $\alpha \leq r \leq \beta$, one can write $r = (1-\delta)\alpha + \delta\beta$ where $1 \geq \delta \geq 0$. Then for all $v \in V$,

$$g_v^{[\alpha,\beta]}(r) = (1-\delta)h^\alpha(v) + \delta h^\beta(v) \ .$$

Since $W_{(u,v)}(r)$ is a linear function we can write

$$W_{(u,v)}(r) = (1-\delta)W_{(u,v)}(\alpha) + \delta W_{(u,v)}(\beta) \ .$$

So after re-weighting the weight of the edge (u,v) is

$$(1-\delta)W_{(u,v)}(\alpha) + \delta W_{(u,v)}(\beta) + (1-\delta)h^\alpha(u) + \delta h^\beta(u) - (1-\delta)h^\alpha(v) - \delta h^\beta(v) \ .$$

Now this is non-negative as by the definition of h^β and h^α we know that both $W_{(u,v)}(\beta) + h^\beta(u) - h^\beta(v)$ and $W_{(u,v)}(\alpha) + h^\alpha(u) - h^\alpha(v)$ are non-negative.

We now consider the case when one of α or β is not finite. We will prove it for the case where $\beta = +\infty$. The case $\alpha = -\infty$ follows similarly. Consider the simple weighted graph $G_\infty = (V, E, W_\infty)$ where the weight function W_∞ is defined as: if the weight of the edge e is $W(e) = a_e x + b_e$ then $W_\infty(e) = a_e$.

We run the Johnson's algorithm on the graph G_∞. Let $h^\infty(v)$ denote the weight that Johnson's algorithm associates with the vertex v. Then define the weight function $g_v^{[\alpha,\infty]}$ as

$$g_v^{[\alpha,\infty]}(x) = h^\alpha(v) + (x-\alpha)h^\infty(v) \ .$$

We need to prove that for every $\alpha \leq r$ and for every $(u,v) \in V$,

$$W_{(u,v)}(r) + g_u^{[\alpha,\infty]}(r) - g_v^{[\alpha,\infty]}(r) = W_{(u,v)}(r) + h^\alpha(u) + (r-\alpha)h^\infty(u) - h^\alpha(v) - (r-\alpha)h^\infty(v) \geq 0 \ .$$

Let $r = \alpha + \delta$ where $\delta \geq 0$. By the linearity of W we can write $W_{(u,v)}(r) = W_{(u,v)}(\alpha) + \delta a_{(u,v)}$, where $W_{(u,v)}(r) = a_{(u,v)} r + b_{(u,v)}$. So the above inequality can be restated as

$$W_{(u,v)}(\alpha) + \delta a_{(u,v)} + h^\alpha(u) + \delta h^\infty(u) - h^\alpha(v) - \delta h^\infty(v) \geq 0 \ .$$

This now follows from the fact that both $W_{(u,v)}(\alpha) + h^\alpha(u) - h^\alpha(v)$ and $a_{(u,v)} + h^\infty(u) - h^\infty(v)$ are non-negative.

Since the running time of the reweighing part of Johnson's algorithm takes $O(mn)$ time, the overall running time of computing the functions $g_v^{[\alpha,\beta]}$ is $O(mn)$, as claimed.

3 Proof of Theorem 2

In this section we construct a parametric algorithm that computes the distance $\delta(u,v)$ between a given pair of vertices. If one is interested in the actual path realizing this distance, then it can be found with some extra book-keeping that we omit in the proof.

The processing algorithm will output the following advice: for any pair $(u, v) \in V \times V$ the advice consists of a set of increasing real numbers $-\infty = b_0 < b_1 < \cdots < b_t < b_{t+1} = \infty$ and an ordered set of degree-d polynomials $p_0, p_1, \ldots, p_t$, such that for all $b_i \leq r \leq b_{i+1}$ the weight of a shortest path in $G(r)$ from u to v is $p_i(r)$. Note that each p_i corresponds to the weight of a path from u to v. Thus if we are interested in computing the exact path then we need to keep track of the path corresponding to each p_i.

Given r, the instantiation algorithm has to find the i such that $b_i \leq r \leq b_{i+1}$ and then output $p_i(r)$. So the output algorithm runs in time $O(\log t)$. To prove our result we need to show that for any $(u, v) \in V \times V$ we can find the advice in time $O(f(d)n)^{\log n}$. In particular this will prove that $t = O(dn)^{\log n}$ and hence the result will follow.

Definition 1. *A minBase is a sequence of increasing real numbers $-\infty = b_0 < b_1 < \cdots < b_t < b_{t+1} = \infty$ and an ordered set of degree-d polynomials $p_0, p_1, \ldots, p_t$, such that for all $b_i \leq r \leq b_{i+1}$ and all $j \neq i$, $p_i(r) \leq p_j(r)$.*

We call the sequence of real numbers the *breaks*. We call each interval $[b_i, b_{i+1}]$ the i-th interval of the minBase and the polynomial p_i the i-th polynomial. The *size* of the minBase is t.

The final advice that the preprocessing algorithm produces is a minBase for every pair $(u, v) \in V \times V$ where the i-th polynomial has the property that $p_i(r)$ is the distance from u to v in $G(r)$ for each $b_i \leq r \leq b_{i+1}$.

Definition 2. *A $minBase^\ell(u, v)$ is a minBase corresponding to the ordered pair u, v, where the i-th polynomial p_i has the property that for $r \in [b_i, b_{i+1}]$, $p_i(r)$ is the length of a shortest path from u to v in $G(r)$, that is taken among all paths that use at most 2^ℓ edges.*
A $minBase^\ell(u, w, v)$ is a minBase corresponding to the ordered triple (u, w, v) where the i-th polynomial p_i has the property that for each $r \in [b_i, b_{i+1}]$, $p_i(r)$ is the sum of the lengths of a shortest path from u to w in $G(r)$, among all paths that use at most 2^ℓ edges, and a shortest path from w to v in $G(r)$, among all paths that use at most 2^ℓ edges.

Note that in both of the above definitions some of the polynomials can be $+\infty$ or $-\infty$.

Definition 3. *If B_1 and B_2 are two minBases (not necessarily of the same size), with polynomials p^1_i and p^2_j, we say that another minBase with breaks b'_k and polynomials p'_k is $\min(B_1 + B_2)$ if the following holds.*

1. *For all k there exist i, j such that $p'_k = p^1_i + p^2_j$, and*
2. *For $b'_k \leq r \leq b'_{k+1}$ and for all i, j we have $p'_k(r) \leq p^1_i(r) + p^2_j(r)$.*

Definition 4. *If $B_1, B_2, \ldots, B_s$ are s minBases (not necessarily of the same size), with polynomials $p^1_{i_1}, p^2_{i_2}, \ldots, p^s_{i_s}$, another minBase with breaks b'_k and polynomials p'_k is $\min\{B_1, B_2, \ldots, B_s\}$ if the following holds.*

1. *For all k there exist q such that $p'_k = p^q_{i_q}$, and*
2. *For $b'_k \leq r \leq b'_{k+1}$ and for all $1 \leq q \leq s$ and all i_q, we have $p'_k(r) \leq p^q_{i_q}(r)$.*

Note that using the above definition we can write the following two equations:

$$minBase^{\ell+1}(u, v) = \min_{w \in V} \left\{ minBase^\ell(u, w, v) \right\} \ . \tag{1}$$

$$minBase^\ell(u, w, v) = min \left(minBase^\ell(u, w) + minBase^\ell(w, v) \right) \ . \tag{2}$$

The following claim will prove the result. The proof of the claim is in the Appendix, Section 6.

Claim 1. If B_1 and B_2 are two minBases of sizes t_1 and t_2 respectively, then

(a) $\min(B_1 + B_2)$ can be computed from B_1 and B_2 in time $O(t_1 + t_2)$.
(b) $\min\{B_1, B_2\}$ can be computed from B_1 and B_2 in time $O(f(d)(t_1+t_2))$, where $f(d)$ is the time required to compute the intersection points of two degree-d polynomials. The size of $\min\{B_1, B_2\}$ is $O(d(t_1 + t_2))$.

In order to compute $\min\{B_1, \ldots, B_s\}$ one recursively computes $X = \min\{B_1, \ldots, B_{s/2}\}$ and $Y = \min\{B_{s/2+1}, \ldots, B_s\}$ and then takes takes $\min\{X, Y\}$.

If there are no negative cycles, then the advice that the instantiation algorithm needs from the preprocessing algorithm consists of $minBase^{\lceil \log n \rceil}(u, v)$. To deal with negative cycles, both $minBase^{\lceil \log n \rceil}(u, v)$ and $minBase^{\lceil \log n \rceil + 1}(u, v)$ are produced, and the instantiation algorithm compares them. if they are not equal, then the correct output is $-\infty$.

Also note that $minBase^0(u, v)$ is the trivial minBase where the breaks are $-\infty$ and $+\infty$ and the polynomial is weight $W((u, v))$ associated to the edge (u, v) if $(u, v) \in E$ and $+\infty$ otherwise.

If the size of $minBase^{\ell}(u, v)$ is s_ℓ, then by (1), (2), and by Claim 1 the time to compute $minBase^{\ell+1}(u, v)$ is $O(f(d))^{\log n} s_\ell$ and the size of $minBase^{\ell+1}(u, v)$ is $O(d)^{\log n} s_\ell$. Thus one can compute the advice for u and v in time

$$(O(f(d))^{\log n})^{\log n} = O(n^{(O(1)+\log f(d)) \log n}) ,$$

and the length of the advice string is $O(n^{(O(1)+\log d) \log n})$.

4 Proof of Theorem 3

Given the linear-weighted graph $G = (V, E, W)$, our preprocessing phase begins by verifying that for all $r \in [\alpha, \beta]$, $G(r)$ has no negative weight cycles. From the proof of Lemma 2 we know that this holds if and only if both $G(\alpha)$ and $G(\beta)$ have no negative weight cycles. This, in turn, can be verified in $O(mn)$ time using the Bellman-Ford algorithm. We may now assume that $G(r)$ has no negative cycles for any $r \in [\alpha, \beta]$. Moreover, since our preprocessing algorithm will solve a large set of shortest path problems, each of them on a specific instantiation of G, we will first compute the reweighing functions $g_v^{[\alpha,\beta]}$ of Lemma 2 which will enable us to apply, in some cases, algorithms that assume nonnegative edge weights. Recall that by Lemma 2, the functions $g_v^{[\alpha,\beta]}$ for all $v \in V$ are computed in $O(mn)$ time.

The advice constructed by the preprocessing phase is composed of two distinct parts, which we respectively call the *crude-short* advice and the *refined-long* advice. We now describe each of them.

For each edge $e \in E$, the weight is a linear function $w_e = a_e + x b_e$. Set $K = 8(\beta - \alpha) \max_e |a_e|$. Let $N_0 = \lceil K\sqrt{n} \ln n/\epsilon \rceil$ and let $N_1 = \lceil Kn/\epsilon \rceil$. For $i = 0, 1$ we will use N_i to define $N_i + 1$ points in $[\alpha, \beta]$ and solve certain variants of shortest path problems instantiated in these points.

Consider first the case of splitting $[\alpha, \beta]$ into N_0 intervals. Let $\rho_0 = (\beta - \alpha)/N_0$ and consider the points $\alpha + i\rho_0$ for $i = 0, \ldots, N_0$. The crude-short part of the preprocessing algorithm solves $N_0 + 1$ *limited* all-pairs shortest path problems in $G(\alpha + i\rho_0)$ for $i = 0, \ldots, N_0$. Set $t = 4\sqrt{n} \ln n$, and let $d_i(u, v)$ denote the length of a shortest path from u to v in $G(\alpha + i\rho_0)$ that is chosen among all paths containing at most t vertices (possibly $d_i(u, v) = \infty$ if no such path exists). Notice that $d_i(u, v)$ is not necessarily the distance from u to v in $G(\alpha + i\rho_0)$, since the latter may require more than t vertices. It is straightforward to compute shortest paths limited to at most k vertices (for any $1 \le k \le n$) in a real-weighted directed graph with n vertices in time $O(n^3 \log k)$ time, by the repeated squaring technique. In fact, they can be computed in $O(n^3)$ time (saving the $\log k$ factor) using the method from [1], pp. 204–206. This algorithm also constructs the predecessor data structure that represents the actual paths. It follows that for each ordered pair of vertices u, v and for each $i = 0, \ldots, N_0$, we can compute $d_i(u,v)$ and a path $p_i(u,v)$ yielding $d_i(u,v)$ in $G(\alpha + i\rho_0)$ in $O(n^3|N_0|)$ time which is

$$O(n^{3.5} \ln n) \ .$$

We also maintain, at no additional cost, linear functions $f_i(u,v)$ which sum the linear functions of the edges of $p_i(u,v)$. Note also that if $d_i(u,v) = \infty$ then $p_i(u,v)$ and $f_i(u,v)$ are undefined.

Consider next the case of splitting $[\alpha, \beta]$ into N_1 intervals. Let $\rho_1 = (\beta - \alpha)/N_1$ and consider the points $\alpha + i\rho_1$ for $i = 0, \ldots, N_1$. However, unlike the crude-short part, the refined-long part of the preprocessing algorithm cannot afford to solve an all-pairs shortest path algorithm for each $G(\alpha + i\rho_1)$, as the overall running time will be too large. Instead, we randomly select a set $H \subset V$ of (at most) $\sqrt{n}$ vertices. H is constructed by performing $\sqrt{n}$ independent trials, where in each trial, one vertex of V is chosen to H uniformly at random (notice that since the same vertex can be selected to H more than once $|H| \le \sqrt{n}$). For each $h \in H$ and for each $i = 0, \ldots, N_1$, we solve the single source shortest path problem in $G(\alpha + i\rho_1)$ from h, and also (by reversing the edges) solve the single-destination shortest path *toward* h. Notice that by using the reweighing functions $g_v^{[\alpha,\beta]}$ we can solve all of these single source problems using Dijkstra's algorithm. So, for all $h \in H$ and $i = 0, \ldots, N_1$ the overall running time is

$$O(|N_1||H|(m + n \log n)) = O(n^{1.5}m + n^{2.5} \log n) = O(n^{3.5}) \ .$$

We therefore obtain, for each $h \in H$ and for each $i = 0, \ldots, N_1$, a shortest path tree $T_i(h)$, together with distances $d_i^*(h,v)$ from h to each other vertex $v \in V$, which is the distance from h to v in $G(\alpha + i\rho_1)$. We also maintain the functions $f_i^*(h,v)$ that sum the linear equations on the path in $T_i^*(h)$ from h to v. Likewise, we obtain a "reversed" shortest path tree $S_i^*(h)$, together with distances $d_i^*(v,h)$ from each $v \in V$ to h, which is the distance from v to h in $G(\alpha + i\rho_1)$. Similarly, we maintain the functions $f_i^*(v,h)$ that sum the linear equations on the path in $S_i^*(h)$ from v to h.

Finally, for each ordered pair of vertices u, v and for each $i = 0, \ldots, N_1$ we compute a vertex $h_{u,v,i} \in H$ which attains

$$\min_{h \in H} d_i^*(u,h) + d_i^*(h,u) \ .$$

Notice that the time to construct the $h_{u,v,i}$ for all ordered pairs u, v and for all $i = 0, \ldots, N_1$ is $O(n^{3.5})$. This concludes the description of the preprocessing algorithm. Its overall runtime is thus $O(n^{3.5} \ln n)$.

We now describe the instantiation phase. Given $u, v \in V$ and $r \in [\alpha, \beta]$ we proceed as follows. Let i be the index for which the number of the form $\alpha + i\rho_0$ is closest to r. As we have the advice $f_i(u,v)$, we let $w_0 = f_i(u,v)(r)$ (recall that $f_i(u,v)$ is a function). Likewise, let j be the index for which the number of the form $\alpha + j\rho_1$ is closest to r. As we have the advice $h = h_{u,v,j}$, we let $w_1 = f_j^*(u,h)(r) + f_j^*(h,u)(r)$. Finally, our answer is $z = \min\{w_0, w_1\}$. Clearly, the instantiation time is $O(1)$. Notice that if we also wish to output a path of weight z in $G(r)$ we can easily do so by using either $p_i(u,v)$, in the case where $z = w_0$ or using $S_j^*(h)$ and $T_j^*(h)$ (we take the path from u to h in $S_j^*(h)$ and concatenate it with the path from h to v in $T_j^*(h)$) in the case where $z = w_1$.

It remains to show that, with very high probability, the result z that we obtain from the instantiation phase is at most ϵ larger than the distance from u to v in $G(r)$. For this purpose, we first need to prove that the random set H possesses some "hitting set" properties, with very high probability.

For every pair of vertices u and v and parameter r, let $p_{u,v,r}$ be a shortest path in $G(r)$ among all simple paths from u to v containing at least $t = 4\sqrt{n} \ln n$ vertices (if G is strongly connected then such a path always exist, and otherwise we can just put $+\infty$ for all u, v pairs for which no such path exists). The following simple lemma is used in an argument similar to one used in [16].

Lemma 3. *For fixed u, v and r, with probability at least $1 - o(1/n^3)$ the path $p_{u,v,r}$ contains a vertex from H.*

Proof. Indeed, the path from $p_{u,v,r}$ by its definition has at least $4\sqrt{n}\ln n$ vertices. The probability that all of the $\sqrt{n}$ independent selections to H failed to choose a vertex from this path is therefore at most

$$\left(1-\frac{4\sqrt{n}\ln n}{n}\right)^{\sqrt{n}} < e^{-4\ln n} < \frac{1}{n^4} = o(1/n^3)\ .$$

Let us return to the proof of Theorem 3. Suppose that the distance from u to v in $G(r)$ is δ. We will prove that with probability $1-o(1)$, H is such that for every u, v and r we have $z \le \delta+\epsilon$ (clearly $z \ge \delta$ as it is the precise length of some path in $G(r)$ from u to v). Assume first that there is a path p of length δ in $G(r)$ that uses less than $4\sqrt{n}\ln n$ edges. Consider the length of p in $G(\alpha+i\rho_0)$. When going from r to $\alpha+i\rho_0$, each edge e with weight a_ex+b_e changed its length by at most $|a_e|\rho_0$. By the definition of K, this is at most $\rho_0K/(8(\beta-\alpha))$. Thus, p changed its weight by at most

$$(4\sqrt{n}\ln n)\cdot\rho_0\frac{K}{8(\beta-\alpha)} = (4\sqrt{n}\ln n)\frac{K}{8N_0} < \frac{\epsilon}{2}.$$

It follows that the length of p in $G(\alpha+i\rho_0)$ is less than $\delta+\epsilon/2$. But $p_i(u,v)$ is a shortest path from u to v in $G(\alpha+i\rho_0)$ of all the paths that contain at most t vertices. In particular, $d_i(u,v) \le \delta+\epsilon/2$. Consider the length of $p_i(u,v)$ in $G(r)$. The same argument shows that the length of $p_i(u,v)$ in $G(r)$ changed by at most $\epsilon/2$. But $w_0 = f_i(u,v)(r)$ is that weight, and hence $w_0 \le \delta+\epsilon$. In particular, $z \le \delta+\epsilon$.

Assume next that every path of length δ in $G(r)$ uses at least $4\sqrt{n}\ln n$ edges. Let p be one such path. When going from r to $r' = \alpha+j\rho_1$, each edge e with weight a_ex+b_e changed its length by at most $|a_e|\rho_1$. By the definition of K, this is at most $\rho_1K/(8(\beta-\alpha))$. Thus, p changed its weight by at most

$$n\cdot\rho_1\frac{K}{8(\beta-\alpha)} = n\frac{K}{8N_1} < \frac{\epsilon}{8}.$$

In particular, the length of $p_{u,v,r'}$ is not more than the length of p in $G(r')$, which, in turn, is at most $\delta+\epsilon/8$. By Lemma 3, with probability $1-o(1/n^3)$, some vertex of h appears on $p_{u,v,r'}$. Moreover, by the union bound, with probability $1-o(1)$ *all* paths of the type $p_{u,v,r'}$ (remember that r' can hold one of $O(n)$ possible values) are thus covered by the set H. Let h' be a vertex of H appearing in $p_{u,v,r'}$. We therefore have $d^*_j(u,h')+d^*_j(h',v) \le \delta+\epsilon/8$. Since $h = h_{u,v,j}$ is taken as the vertex which minimizes these sums, we have, in particular, $d^*_j(u,h)+d^*_j(h,v) \le \delta+\epsilon/8$. Consider the path q in $G(\alpha+j\rho_1)$ realizing $d^*_j(u,h)+d^*_j(h,v)$. The same argument shows that the length of q in $G(r)$ changed by at most $\epsilon/8$. But $w_1 = f^*_j(u,h)(r)+f^*_j(h,v)(r)$ is that weight, and hence $w_1 \le \delta+\epsilon/4$. In particular, $z \le \delta+\epsilon/4$.

5 Concluding remarks

We have constructed several parametric shortest path algorithms, whose common feature is that they preprocess the generic instance and produce an advice that enables particular instantiations to be solved faster than running the standard weighted distance algorithm from scratch. It would be of interest to improve upon any of these algorithms, either in their preprocessing time or in their instantiation time, or both.

Perhaps the most challenging open problem is to improve the preprocessing time of Theorem 2 to a polynomial one, or, alternatively, prove an hardness result for this task. Perhaps less ambitious is the preprocessing time in Theorem 1. The only bottleneck that prevents reducing the $\tilde{O}(n^4)$ time to $O(nm)$ is the need to solve the LP using interior point methods in order to compute the range of non-negative cycles. Perhaps this could be circumvented.

Finally, parametric algorithms are of practical importance for other combinatorial optimization problems as well. It would be interesting to find applications where, indeed, a parametric algorithm can be truly beneficial, as it is in the case of shortest path problems.

Acknowledgment

We thank Oren Weimann and Shay Mozes for useful comments.

References

1. A. V. Aho, J. E. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Longman Publishing Co., Boston, MA, 1974.
2. R. Bellman, *On a routing problem*, Quarterly of Applied Mathematics 16 (1958), 87–90.
3. P. Carstensen, *The complexity of some problems in parametric linear and combinatorial programming*, Ph.D. Thesis, Mathematics Dept., U. of Michigan, Ann Arbor, Mich., 1983.
4. T. M. Chan, *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*, Proceedings of the 39^{th} ACM Symposium on Theory of Computing (STOC), ACM Press (2007), 590–598.
5. D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation 9 (1990), 251–280.
6. E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematik 1 (1959), 269–271.
7. R. W. Floyd, *Algorithm 97: shortest path* Communications of the ACM 5 (1962), 345.
8. M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM Journal on Computing 5 (1976), 49–60.
9. M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM 34 (1987), 596–615.
10. D. Gusfield *Parametric combinatorial computing and a problem of program module distribution*, Journal of the ACM 30(3) (1983), 551–563.
11. D. B. Johnson, *Efficient algorithms for shortest paths in sparse graphs*, Journal of the ACM 24 (1977), 1–13.
12. N. Karmarkar, *A New Polynomial Time Algorithm for Linear Programming*, Combinatorica 4 (1984), 373-396.
13. R. M. Karp and J. B. Orlin, *Parametric shortest path algorithms for with an application to cycle staffing*, Discrete Applied Mathematics 3 (1981), 37–45.
14. E. Nikolova, J. A. Kelner, M. Brand and M. Mitzenmacher, *Stochastic Shortest Paths Via Quasi-convex Maximization*, Proceedings of the 14^{th} Annual European Symposium on Algorithms (ESA), LNCS (2006), 552–563.
15. N. E. Young, R. E. Tarjan and J. B. Orlin, *Faster parametric shortest path and minimum-balance algorithms*, Networks 21 (1991), 205–221.
16. U. Zwick, *All-pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM 49 (2002), 289–317.

Appendix

6 Proof of Claim 1

The Claim 1 is the following:

Claim 2. If B_1 and B_2 are two minBases of sizes t_1 and t_2 respectively, then

(a) $\min(B_1 + B_2)$ can be computed from B_1 and B_2 in time $O(t_1 + t_2)$.
(b) $\min\{B_1, B_2\}$ can be computed from B_1 and B_2 in time $O(f(d)(t_1 + t_2))$, where $f(d)$ is the time required to compute the intersection points of two degree-d polynomials. The size of $\min\{B_1, B_2\}$ is $O(d(t_1 + t_2))$.

Proof (Proof of Claim 1).
Let $break_1 = \{b^1_0, b^1_1, \dots, b^1_{t_1}, b^1_{t_1+1}\}$ and $break_2 = \{b^2_0, b^2_1, \dots, b^2_{t_2}, b^2_{t_2+1}\}$ be the set of breaks of B_1 and B_2 respectively. Also let $\{p^1_0, p^1_1, \dots, p^1_{t_1}\}$ and $\{p^2_0, p^2_1, \dots, p^2_{t_2}\}$ be the ordered set of polynomials for B_1 and B_2 respectively.

Proof of Part (a): Consider the sequence of breaks whose elements are $break_{1+2} = break_1 \cup break_2$ after sorting. Let $break_{1+2} = \{b^{1+2}_0, b^{1+2}_1, \dots, b^{1+2}_{t_1+t_2}, b^{1+2}_{t_1+t_2+1}\}$. Now for an interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ there exists a $p^1_{i_1}$ such that $p^1_{i_1} \le p^1_j$ for all j. Similarly for the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ there exists a $p^2_{i_2}$

such that $p^2_{i_2} \leq p^2_j$ for all j. Thus in the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ the polynomial $p^1_{i_1} + p^2_{i_2}$ is less than or equal to $p^1_j + p^2_k$ for all j, k. Thus $break^{1+2}$ can be a set of breaks corresponding to $\min(B_1 + B_2)$ and $p^1_{i_1} + p^2_{i_2}$ is the polynomial corresponding to the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$. Thus $\min(B_1 + B_2)$ can be computed in time $O(t_1 + t_2)$.

Proof of Part (b): Just like in the proof of Part (a) consider the sequence of breaks whose elements are $break_{1+2} = break_1 \cup break_2$. Let $break_{1+2} = \{b^{1+2}_0, b^{1+2}_1, \ldots, b^{1+2}_{t_1+t_2}, b^{1+2}_{t_1+t_2+1}\}$. Now for an interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ there exists a $p^1_{i_1}$ such that $p^1_{i_1} \leq p^1_j$ for all j. Similarly for the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ there exists a $p^2_{i_2}$ such that $p^2_{i_2} \leq p^2_j$ for all j.

Now in the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ the two polynomials $p^1_{i_1}$ and $p^2_{i_2}$ can intersect each other at most d times, because they are degree d-polynomials. Let $b^{i_1,i_2}_1 < b^{i_1,i_2}_2 < \cdots < b^{i_1,i_2}_c$ for $c \leq d$ be the set of points where $p^1_{i_1}$ and $p^2_{i_2}$ intersect within the interval $[b^{1+2}_i, b^{1+2}_{i+1}]$. Thus in the subintervals $[b^{i_1,i_2}_k, b^{i_1,i_2}_{k+1}]$ either $p^1_{i_1}$ or $p^2_{i_2}$ is smaller and they are smaller than all other polynomials p^1_j and p^2_j in this interval. Thus $break_{1+2}$ along with all the intersection points of $p^1_{i_1}$ and $p^2_{i_2}$ in each interval $[b^{1+2}_i, b^{1+2}_{i+1}]$ can be a set of breaks for $\min\{B_1, B_2\}$ and the polynomials can also be computed easily. So $\min\{B_1, B_2\}$ can be computed in time $O(f(d)(t_1 + t_2))$ where $f(d)$ is the time taken to compute the intersection points of two degree d polynomials. The size of $\min\{B_1, B_2\}$ is $O(d(t_1 + t_2))$.