Floats, Integers, and Single Source Shortest Paths*

Mikkel Thorup
Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
mthorup@diku.dk, http://www.diku.dk/~mthorup

Abstract

Floats are ugly, but to everyone but theoretical computer scientists, they are the real thing. A linear time algorithm is presented for the undirected single source shortest paths problem with positive floating point weights.

1 Introduction

The technical goal of this paper is to present a linear time solution to the undirected single source shortest paths problem (USSSP) where the weights are positive floating points, or just floats. On a more philosophical level, the goal is to draw attention to the problem of making efficient algorithms for floats. Suppose, for example, we have an algorithm for the max-flow problem whose running time includes a factor $\log C$, where C is the maximal capacity. If we allow floating points, such an algorithm is not even polynomial, i.e. $\log C$ is the exponent of C, and the exponent is stored with $\log \log C$ bits.

Floating points are at least as used as integers, by everybody but theoretical computer scientists, who seem to prefer integers. To multiply two integers the faster way is often to convert them to double floats and send them to the floating point co-processor. Why? Because people simply don't care enough about integers to make an integer co-processor. Also, as theoretical computer scientists, we have to admit that floats do provide an elegant way of dealing with numbers in a large range. It is for good reasons that all other scientists and engineers have used them for centuries.

The rounding of floating point arithmetic is ugly in that, for example, addition is neither associative, nor commutative. Nevertheless, in this paper, we hope to indicate, that finding the structure of the rounding for a given problem, such as USSSP, can be an appealing challenge.

It should be noted that often floats do not cause any problems relative to integers. For problems like sorting, priority queues, and searching, there is no difference. The IEEE floating point standard is made such that interpreting the bit-string representation of floating points as representing integers, is order preserving. Hence we can feed floating points to an integer priority queue, if we just don't tell it that it is floats. Similarly, van Emde Boas' data

*Most of this work was done while the author visited the Max-Planck-Institut für Informatik.

structure [vBKZ77] works in time $O(\log \omega)$, where ω is the word length, no matter whether the words represent integers or floats.

Up to recently, all theoretical developments in the single source shortest paths problem (SSSP) were based in Dijkstra's algorithm [Dij59], where vertices are visited in increasing order of distance to the source using a priority queue. Since the priority queue doesn't care whether the input is integers or floats, all implementations of Dijkstra's algorithm work equally well for integers and floats. However, recently, the author [Tho97] presented a linear time non-Dijkstra algorithm for undirected SSSP with integer weights. It is crucial to this algorithm that the weights are sorted with respect to their exponents. For integers in words, the exponent is at most ω, and then the exponents are easily sorted in linear time. However, for floating points, sorting the exponents is as hard as integer sorting, and this we do not know how to do in linear time.

Here, we show how to move edges around in the graph, preserving the floating point distances, ending with a series of independent subgraphs for which the exponents can be sorted in linear time. Given the sorted exponents, it is not too difficult to modify the algorithm from [Tho97] for integer USSSP to solve the floating point USSSP in linear time. Some remarks on this will be made in the journal version of [Tho97]. Given the sorted exponents, such a floating point version of the algorithm from [Tho97] may work very well in practice because of the efficiency of floating point arithmetic. However, here we give a self contained presentation, showing something in principle stronger; namely, that USSSP with floating point weights can be solved in linear time using a linear time oracle for USSSP with integer weights. Finally, we will make a few remarks on floats and max-flow.

2 Preliminaries

Our algorithm runs on a RAM, which models what we program in imperative programming languages such as C. The memory is divided into addressable words of length ω. Addresses are themselves contained in words. Hence, in order to address the n different nodes of the graph we are working on, we assume $\omega \geq \log n$. Moreover, we have a constant number of registers, each with capacity for one word. The basic assembler instructions are: conditional jumps, direct and indirect addressing for loading and storing words in registers, and some computational instructions, such as comparisons, addition, and multiplication, for numbers in registers. The space complexity is the maximum memory address used, and the time complexity is the number of instructions performed. All weights are floating point numbers, each contained in $O(1)$ words.

The numbers may be either integers, represented the usual way, of floating point numbers. A *floating point*, or just a *float*, is a pair $x = (e, m)$, where e is an integer, and m is a bit string $b_1 \cdots b_\wp$. Then x represents the real number $2^e(1 + \sum_{i=1}^{\wp} b_i/2^i)$. Thus $e = \lfloor \log_2 x \rfloor$. We call e the *exponent*, denoted $\texttt{expo}(x)$, and m the *mantissa*, denoted $\texttt{mant}(x)$. Often we will identify a float with the real number it represents. Both e and m are assumed to fit in a constant number of words. The number $\wp = O(\omega)$ is fixed throughout a given computation and is referred to as the *precision*. We have previously mentioned that according to the IEEE floating point standard, we get the correct ordering of floats by perceiving them as integers. This is obtained by first having the sign-bit of m, then e, and finally m without the sign-bit.

We let $\oplus$ denote floating point addition. In this paper, for simplicity, when two floats x and y are added, they are always rounded *down* to nearest float (determined by the precision

$\wp$). Thus $x \oplus y \le x + y$. This gives us the following basic rule:

$$\texttt{expo}(x) > \texttt{expo}(y) + \wp \;\Rightarrow\; x \oplus y = y \oplus x = x \tag{1}$$

If instead we were rounding either up or down to nearest float, the rule would only apply if $\texttt{expo}(x) > \texttt{expo}(y) + \wp + 1$, and all the calculations below, would have to be changed accordingly.

Let $G = (V, E)$, $|V| = n$, $|E| = m$, be an undirected connected graph with a distinguished source vertex s. Each edge $e \in E$ has a positive floating point weight $\ell(e)$ associated with it. The length of a path from s to some vertex v, is the floating point sum of the weights added up starting from s. More specifically, if the path is $P = (v_0, v_1, \ldots, v_l)$, $s = v_0$, $v_l = v$, then the length of P is

$$((((\ell(v_0, v_1) \oplus \ell(v_1, v_2)) \oplus \ell(v_2, v_3)) \cdots) \oplus \ell(v_{l-1}, v_l)$$

By $d(v)$ we denote the length of the shortest path from s to v. The floating point USSSP problem is that of finding $d(v)$ for all v. Different orders of adding the weights could give different answers, so it is natural to ask how big the errors can get. Let $d^*(v)$ denote the distance where the weights are summed using normal addition of the reals represented by the floats. Then $d^*(v)$ represents the ideal answer. Clearly $d^*(v) \ge d(v)$.

Observation 1 $\frac{d^*(v) - d(v)}{d(v)} \le 2^{-\wp + \log_2 n}$

Proof: Set $e = \texttt{expo}(d(v))$. We are adding at most $n - 1$ numbers. Since the maximal exponent is $\le e$, the maximal loss per number is $< 2^{e-\wp}$. Thus, the total error is $< (n-1)2^{e-\wp} < 2^{e-\wp+\log_2 n}$. ∎

In theory, since $\omega \ge \log_2 n$, we can always simulate $\log_2 n$ extra bit of precision without affecting the asymptotic running time. In practice, according to the IEEE standard format, with long floats, we have $\wp = 52$. Hence the relative error $\frac{d^*(v) - d(v)}{d(v)}$ is at most $2^{-52+\log_2 n}$, which is normally OK. Throughout the paper, we will assume $p > \log_2 n$.

3 Sorting the weight exponents

First, by a *source component* of a graph with source s, we mean a maximal subgraph H with $H \setminus \{s\}$ connected. We will now apply some different transformations to G constructing a graph G' so that the d-values of the nodes are unchanged (though some vertices may be identified), but so that for each source component of G', the weight exponents vary by at most $n(\wp + 1)$. Since the source components only intersect in s, we can solve the USSSP problem independently for each source component H. The small variation in the weight exponents within H implies that they can be sorted in linear time.

Our first step is to construct a minimum spanning tree T for G in deterministic linear time and space [FW94]. The algorithm from [FW94] assumes integer weights, but since we are only interested in the ordering, we can just perceive our floats as representing integers when running the algorithm from [FW94].

Let $d_T(v)$ denote the weight of the path from s to v in T. Let $f_T(v)$ denote the weight of the first edge on this path. Clearly $d_T(v) \ge d(v)$. Also, since T is a minimum spanning tree, $f_T(v) \le d(v)$. Using these simple bounds on $d(v)$, the algorithm below identifies places where the graph can be simplified without affecting the distance to any vertex.

Algorithm A: Transforms G to a graph G' so that the d-values of the nodes are unchanged, though some vertices may be identified, and so that for each source component of G', the weight exponents vary by at most $n(\wp+1)$.

A.1. As described in [FW94], compute a minimum spanning tree T of G.

A.2. Compute $d_T(\cdot)$.

A.3. For all $(v,w) \in E$,

A.3.1. If $v \neq s$, $\mathtt{expo}(\ell(v,w)) > \mathtt{expo}(d_T(v)) + \wp$ and $d_T(w) \geq d_T(v)$, replace (v,w) by an edge (s,w) of the same weight. If (v,w) was in T, it is replaced by (s,w) in T.

A.4. Compute $f_T(\cdot)$

A.5. For all $(v,w) \in T$ with v nearest s and $v \neq s$,

A.5.1. If $(v,w) \in T$ and $\mathtt{expo}(\ell(v,w)) < \mathtt{expo}(f_T(v)) - \wp$, $\ell(v,w) := 0$

A.6. Contract all components induced by edges of weight 0, removing any loops created.

As stated above, the algorithm may introduce multiple edges when adding an edge (s,w) in step A.3.1. However, really we will only have the lightest edge (s,w) in the graph. For each vertex w, we put an edge to s first in the incidence list of w. Then, when a new edge (s,w) is introduced, it is easily compared with the previous one, if any.

Algorithm A is analysed in the remainder of this section. In Lemmas 2–4, we will first prove that Algorithm A does not change any d-values, that T is maintained as a minimum spanning tree, that Algorithm A runs in linear time and space, and that for each edge (v,w), $v \neq s$ in the produced graph G',

(i) $\mathtt{expo}(\ell(v,w)) \leq \mathtt{expo}(d_T(v)) + \wp$

(ii) $\mathtt{expo}(\ell(v,w)) \geq \mathtt{expo}(f_T(v)) - \wp$.

Then in Lemmas 5–8, we will show that (i) and (ii) imply the desired bounded variance of $n(\wp+1)$ on the weight exponents within each source component of G'. Finally, in Lemma 9, we will show that the bounded variance on the weight exponents of a source component imply that they can be sorted in linear time.

Lemma 2 *During the course of Algorithm A, all d-values remain unchanged, and all d_T- and f_T values remain unchanged after once computed. Finally T is maintained as a minimum spanning tree.*

Proof: We need to argue that steps A.3.1, A.5.1, and A.6 cannot violate the claim of the lemma. First, suppose (v,w) pass the test of step A.3.1. Then $\mathtt{expo}(\ell(v,w)) > \mathtt{expo}(d_T(v)) + \wp \geq \mathtt{expo}(d(v)) + \wp$. Hence, by (1), $d(v) \oplus \ell(v,w) = \ell(v,w)$ and $d(v)_T \oplus \ell(v,w) = \ell(v,w)$. Also, $d_T(w) \oplus \ell(v,w) \geq d(w) \oplus \ell(v,w) \geq \ell(v,w) > d_T(v) \geq d(v)$, so no shortest path in G or T comes to v via (w,v). Thus, replacing (v,w) by (s,w) with $\ell(s,w) = \ell(v,w)$ in G and in T does not change any d- or d_T-value. In contrast, step A.3.1 may change f_T-values, and this is why the f_T values are not computed until step A.4.

Concerning the maintainance of T as a minimum spanning tree in step A.3.1, suppose first that $(v,w) \in T$. Since $d_T(v) \leq d_T(w)$, this implies that v is nearest to s; for otherwise, $d_T(v) = d_T(w) \oplus \ell(v,w) \geq \ell(v,w) > d_T(v)$. Thus, deleting (v,w) cuts T in two components with s in the same component as v. We know that (v,w) was a minimal edge crossing this cut, and since (s,w) crosses the same cut with the same weight, we get a new minimal spanning tree when replacing (v,w) by (s,w) in T. Now, if (v,w) was not in T, we know that $\ell(v,w)$ was at least as big as any weight on the path from v to w in T. Then $\ell(s,w) = \ell(v,w)$ is also bigger than all the weights on the path from s to w, for all edges on this path that are not on the path from v to w are on the path from s to v, and all these edges have weight $\leq d_T(v) < \ell(v,w) = \ell(s,w)$. Thus (s,w) is at least as heavy as any edge on the path from s to w, and hence we do not need to involve it in T. Thus, we conclude that step A.3.1 cannot violate the lemma.

Suppose (v,w) pass the test of step A.5.1 and gets its weight decreased to 0. Since $(v,w) \in T$, T remains a minimum spanning tree. Since $\texttt{expo}(\ell(v,w)) < \texttt{expo}(f_T(v)) - \wp$ and $d(v) \geq f_T(v)$, $d(v) \oplus \ell(v,w) = d(v)$, so $d(w) \leq d(v)$. Also, $d_T(v) \geq f_T(v)$, so $d_T(w) \leq d_T(v)$. However, $f_T(v) = f_T(w)$, so symmetrically, $d(v) \leq d(w)$ and $d_T(v) \leq d_T(w)$. Thus $d(v) = d(w)$ and $d_T(v) = d_T(w)$, and hence it does not change any d- or d_T-value to set $\ell(v,w) := 0$. Further, since the structure of T is unchanged and $v \neq s$, no f_T-value is changed by step A.5.1.

Concerning the contraction in step A.6, note that all 0 weight edges are in T and not incident to s. Hence, the contraction of 0 weight edges does not change any d-, d_T-, or f_T-values, and neither does the contraction of edges in T stop T from being a minimal spanning tree. ■

Lemma 3 *Algorithm A runs in linear time and space.*

Proof: Step A.1 is known to take $O(m)$ time and space [FW94]. Steps A.2 and A.4 are done in time $O(n)$, each by a simple scan from starting in the root. By Lemma 2, we do not need to worry about updating d_T- and f_T-values when once computed, and hence it is straightforward to implement each of step A.3.1 and step A.5.1 in constant time. Hence step A.3 takes $O(m)$ time and step A.5 takes $O(n)$ time. For step A.6, we identify the components of the graph induced by 0 weight edges in time $O(m)$. Then we map the nodes of each component to a surviving representative of that component. Finally, we contract the components in $O(m)$ total time and space by renaming the end-points according to the map, removing all loops created. ■

Lemma 4 *All edges $(v,w) \in G'$ with $v \neq s$ satisfy (i) and (ii).*

Proof: By Lemma 2, d_T- and f_T-values are not changed once computed. Hence, no edge in the resulting graph G' satisfy the condition of step A.3.1 or of step A.5.1.

Let $(v,w) \in G'$, $v \neq s$. Suppose for a contradiction that (v,w) violates (i), that is, $\texttt{expo}(\ell(v,w)) > \texttt{expo}(d_T(v)) + \wp$. If $d_T(w) \geq d_T(v)$, (v,w) should have passed the test of step A.3.1, contradicting that $(v,w) \in G'$. However, if $d_T(w) < d_T(v)$, we have $\texttt{expo}(\ell(v,w)) > \texttt{expo}(d_T(v)) + \wp > \texttt{expo}(d_T(w)) + \wp$, so then (w,v) should have passed the test of step A.3.1. Thus, we conclude that (i) is not violated.

Suppose instead that (v,w) violates (ii), that is, $\texttt{expo}(\ell(v,w)) < \texttt{expo}(f_T(v)) - \wp$. If $(v,w) \in T$ with v nearest to s, (v,w) should have passed the test of step A.5.1, contradicting $(v,w) \in G'$. If instead w is nearest to s and $w \neq s$, (w,v) should have passed the test since $f_T(v) = f_T(w)$. Finally, we cannot have $(w,v) \in T$ and $w = s$, for then $\texttt{expo}(\ell(v,w)) = \texttt{expo}(f_T(v))$, contradicting $\texttt{expo}(\ell(v,w)) < \texttt{expo}(f_T(v)) - \wp$.

Now, suppose $(v,w) \in G' \setminus T$. Since T is a minimum spanning tree, the weight of $\ell(v,w)$ is at least as big as the weight of any edge on the path from v to w in T. Consider the path P from v to w in T. If $s \in P$, let (s,u) be the first edge on the path from s to v in P.

Then $f_T(v) = \ell(s,u)$, which together with $\texttt{expo}(\ell(v,w)) < \texttt{expo}(f_T(v)) - \wp < \texttt{expo}(f_T(v))$ contradicts that $\ell(v,w)$ is as big as any weight in P. Thus $s \notin P$.

Consider any edge $(x,y) \in P$ with x nearest s. Since $s \notin P$, $f_T(v) = f_T(x)$. Hence, $\texttt{expo}(\ell(x,y)) \le \texttt{expo}(\ell(v,w)) < \texttt{expo}(f_T(v)) - \wp = \texttt{expo}(f_T(x)) - \wp$. Thus, (x,y) satisfies the condition of step A.5.1. Consequently, all edges in P should have been contracted in step A.6, and then (v,w) should have been removed as a loop. This contradicts $(v,w) \in G'$, and completes the proof of the lemma. ■

We can now solve the USSSP problem for G, by solving it independently for each source component of G'. Thus, in the following, let H by any source component of G'. Then $H \setminus \{s\}$ is connected and (i) and (ii) are satisfied for all $(v,w) \in H$ with $v \neq s$. In Lemmas 5–8, we will prove that these properties suffice for a bounded variation of $n(\wp+1)$ on the weight exponents within H.

Lemma 5 *If $v \in H$ has depth $d > 0$, $\texttt{expo}(d_T(v)) \le \texttt{expo}(f_T(v)) + (d-1)(\wp+1)$.*

Proof: (induction on d) The statement is trivially true for $d = 1$. If $d > 1$ and u is the ancestor of v, by induction, $\texttt{expo}(d_T(u)) \le \texttt{expo}(f_T(u)) + (d-2)(\wp+1) = \texttt{expo}(f_T(v)) + (d-2)(\wp+1)$. Applying (i), $\texttt{expo}(\ell(u,v)) \le \texttt{expo}(d_T(u)) + \wp \le \texttt{expo}(f_T(v)) + (d-1)(\wp+1) - 1$. Since $d_T(v) = d_T(u) \oplus \ell(u,v)$, $\texttt{expo}(d_T(v)) \le \max\{d_T(u), \ell(u,v)\} + 1 \le \texttt{expo}(f_T(v)) + (d-1)(\wp+1)$. ■

Lemma 6 *If $(v,w) \in H$ where $v \neq s$, $\texttt{expo}(\ell(v,w)) \le \texttt{expo}(f_T(v)) + depth_T(v)(\wp+1) - 1$.*

Proof: By (i) and Lemma 5, $\texttt{expo}(\ell(v,w)) \le \texttt{expo}(d_T(v)) + \wp \le \texttt{expo}(f_T(v)) + (depth_T(v))(\wp+1) + \wp$. ■

Lemma 7 *If (s,u) and (v,w) are in H, $\texttt{expo}(\ell(v,w)) \le \texttt{expo}(\ell(s,u)) + (|V(H)|-1)(\wp+1) - 1$.*

Proof: Let $T_1, \dots, T_i$ be distinct trees in $T \setminus \{s\}$ such that $u \in T_1$, and for $i > 1$, there is an edge (x,y) from T_{i-1} to T_i. By induction on i, we will argue that if $(a,b) \in H$ and $a \in T_i$,

$$\texttt{expo}(\ell(a,b)) \le \texttt{expo}(\ell(s,u)) + (\sum_{j=1}^{i} |V(T_j)|(\wp+1)) - i. \tag{2}$$

Suppose $i = 1$. Since $depth_T(a) \le |V(T_i)|$, (2) follows directly from Lemma 6.

Suppose $i > 1$. Let (x,y) be the edge entering T_i from T_{i-1}. By induction,

$$\texttt{expo}(\ell(x,y)) \le \texttt{expo}(\ell(s,u)) + (\sum_{j=1}^{i-1} |V(T_j)|(\wp+1)) - (i-1).$$

Since T is a minimum spanning tree $\ell(x,y) \ge f_T(y) = f_T(a)$. Thus, by Lemma 6,

$$\begin{aligned}\texttt{expo}(\ell(a,b)) &\le \texttt{expo}(f_T(a)) + depth_T(a)(\wp+1) - 1\\ &\le \texttt{expo}(\ell(x,y)) + |V(T_i)|(\wp+1) - 1\\ &\le \texttt{expo}(\ell(s,u)) + (\textstyle\sum_{j=1}^{i} |V(T_j)|(\wp+1)) - i,\end{aligned}$$

completing the proof of (2). The trees $T_1, \dots, T_i$ can be chosen for any $(a,b) \in H$, and since $\bigcup_j T_j \subseteq H \setminus \{s\}$, the lemma follows. ■

Lemma 8 *The minimum and maximum weight exponent of an edge in H differ by at most $|V(H)|(\wp+1)$.*

Proof: Let (s, u) be the minimum weight edge in H leaving s. Consider any edge (v, w). By (ii), $\texttt{expo}(\ell(v,w)) \geq \texttt{expo}(f_T(v)) - \wp \geq \texttt{expo}(\ell(s,u)) - \wp$. At the same time, by Lemma 7, $\texttt{expo}(\ell(v,w)) \leq \texttt{expo}(\ell(s,u)) + (|V(H)|-1)(\wp+1) - 1$. Hence, exponents of weights in H can vary by at most $|V(H)|(\wp+1) - 1$. ∎

Lemma 9 *We can sort the weight exponents of the edges in H in linear time and space.*

Proof: First subtracting the minimum weight exponent from all other weight exponents, the maximum weight exponent becomes $n(\wp+1)$, which is represented by $\log_2 n + \log_2(\wp+1)$ bits. If $\wp \leq n-1$, each weight exponent is viewed as two $\leq \log_2 n$ bit characters, and we radix sort in two rounds, each round taking linear time and space.

In the extreme case where $\wp > n$, since $\wp = O(\omega)$, we have $\log_2 n + \log_2 \wp = O(\omega/(\log n \log\log n))$. Then we can apply the linear time and space packed sorting from [AH97]. ∎

Combining Lemma 2, 3, 8, and 9, we conclude,

Theorem 10 *Given a floating point weight graph G with source vertex s, in linear time and space, we can transform G into a graph G' with the same d-values, though possibly with some vertices identified, such that for each source component H, the minimum and maximum weight exponent of an edge in H differ by at most $|V(H)|(\wp+1)$. Further, we can sort the weight exponents of each source component H in linear time and space.*

Concerning a concrete implementation of our algorithm, it should be noted that it would suffice with the minimal spanning tree T just being minimal with respect to the exponents of the weights. Dealing just with the exponents rather than the whole floats may be easier. For example, for 64-bit floats in the IEEE floating point standard, the exponent an 11-bit integer, or really, the exponent is represented as an unsigned 11-bit integer from which one has to subtract 1023 to get the exponent. For the ordering of the exponents, however, it suffices just to mask out the 11 exponent bits. Also, in practice, one would certainly prefer the randomized linear time algorithm of Karger, Klein, and Tarjan [KKT95] to the deterministic one of Fredman and Willard [FW94].

4 Using an integer oracle

Having applied Theorem 10, we complete solving the USSSP problem for G, by solving it in $O(|E(H)|)$ time and space for each of the source components H of G'. In our solution, we will apply the linear time integer USSSP oracle from [Tho97]. First we will estimate the exponents ± 1. Second we will make exact calculations.

Distances with a bit too much precision

We will calculate the distances from s but sometimes using some extra precision. Since we always round down, extra precision means larger values. For each vertex v, we denote the obtained distance by $D(v)$. Referring to Observation 1, we get $d(v) \leq D(v) \leq d^*(v)$.

Since $\wp \geq \log_2 n \geq \log_2 |V(H)|$, we further get $D(v) \leq 2d(v)$. Consequently, $\texttt{expo}(d(v)) \leq \texttt{expo}(D(v)) \leq \texttt{expo}(d(v)) + 1$.

Our first step is to subtract the exponent e of the smallest weight from all weight exponents (corresponding to division by 2^e). By Theorem 10, all weight exponents are now in the interval $[0, n(\wp + 1)) \subseteq [0, 2n\wp)$.

We are going to proceed in rounds for $i = 1, \ldots, 2n$. In round i, we are going to find the distances from s along paths where the maximal exponent of an edge weight is $< i\wp$. We assume that this has already been done over paths where the maximal exponent is $< (i-1)\wp$.

Consider the set $S = \{v \mid \texttt{expo}(D(v)) < (i-1)\wp\}$. Edges with weight exponents $\geq (i-1)\wp$ are not going to give any better distances for vertices $v \in S$, so the distances to vertices in S may now be output. Next we *contract* S as follows. For all edges $(v, w) \in E$, $v \in S \setminus \{s\}$, $w \notin S$, we replace (v, w) by a *distance edge* (s, w) whose weight is $D(v) \oplus \ell(v, w)$. Afterwards $S \setminus \{s\}$ is removed. Clearly the remaining vertices have same distances in the reduced graph, and each edge (v, w) is only once replaced by a distance edge (s, w). Multiple distance edges (s, w) are assumed removed as described immediately below Algorithm A.

Now contract all edges (v, w) with $\texttt{expo}(\ell(v, w)) < (i-2)\wp$. This corresponds to reducing their weight to 0. We claim that this does not not change the floating point distance to any vertex. By symmetry, it suffices to show that it does not change the distance to w. Suppose the shortest path to w goes through v. Since v was not contracted in S, $\texttt{expo}(D(v)) \geq (i-1)\wp$, but this immediately implies $D(v) \oplus \ell(v, w) = D(v)$

Here in round i, concerning the original edges, we restrict our attention to the surviving ones with weight exponent $< i\wp$. Since we have contracted all original edges with weight exponent $< (i-2)\wp$, this implies that edges will be considered in their original form for at most 2 rounds. Concerning distance edges, we restrict our attention to the surviving ones where the maximal edge weight on the corresponding path has exponent $< i\wp$. The weight of these distance edges is then $< (|V(H)|-1)2^{i\wp} < 2^{(i+1)\wp}$, which means that their exponents are $< (i+1)\wp$. At the same time, we have contracted every distance edge whose exponent is $< (i-1)\wp$. Hence distance edges are considered for at most 2 rounds. In conclusion, each edge will be considered for a total of at most 4 times.

Before converting our floats to integers, we subtract $(i-3)\wp$ from every exponent, corresponding to dividing all weights by $2^{(i-3)\wp}$. Since the smallest weight exponent was $\geq (i-2)\wp$, it is now $\wp$, so all weights are now integers. All distances computed in round i, are based on original edges whose weight exponents are now $< i\wp - (i-3)\wp = 3\wp$, hence of weight $< 2^{3\wp}$. Thus, the maximal distance computed in round i is $< (|V(H)| - 1)2^{3\wp} \leq 4\wp$ Thus, we call our integer USSSP, where each integer is represented by $4\wp = O(\omega)$ bits. This takes time linear in the number of edges considered, and since each edge is considered in at most 4 rounds, the total running time over all rounds is linear in the total number of edges. The distances produced are essentially correct, except that in the calculations, we vary between $\wp$ and $4\wp$ bits of precision, where we should really only have had a precision of $\wp$ in every single addition. As pointed out above, this means that the exponents of the calculated distances are either correct, or at most one too large.

Exact results

In order to get the exact floating point distances $d(v)$, as defined in Section 2, we are going to proceed in rounds as above, but working with one exponent e at the time. We will go through the exponents e in increasing order. Recall that the exponents of the edge weights were sorted in Section 3. Inserting exponents of computed distances in the ordering is straightforward; for if $d(w) = d(v) \oplus \ell(v,w)$, $\texttt{expo}(d(w)) \in \{\texttt{expo}(d(v)), \texttt{expo}(d(v))+1, \texttt{expo}(\ell(v,w)), \texttt{expo}(\ell(v,w))+1\}$. For each vertex v, we are going to start with our estimated distance $D(v) \geq d(v)$ computed in the previous subsection. We are going to decrease $D(v)$ to $d(v)$ over several rounds. Recall that when we start $\texttt{expo}(d(v)) \leq \texttt{expo}(D(v)) \leq \texttt{expo}(d(v))+1$.

The goal of round e is to get $D(v) = d(v)$ for all v with $\texttt{expo}(d(v)) \leq e$. Inductively, we assume this has already been achieved for all v with $\texttt{expo}(d(v)) \leq e-1$. Let S be the set of vertices v with $\texttt{expo}(d(v)) \leq e-1$. These vertices are contracted, as in the previous section, that is, each outgoing edge (v,w), $v \in S \setminus \{s\}, w \notin S$, is replaced by a distance edge (s,w) with $\ell(s,w) = d(v) \oplus \ell(v,w)$. Moreover, we set $D(w) = \min\{D(w), D(v) \oplus \ell(v,w)\}$.

Construct the graph G_e with the distance edges (s,v) with $\texttt{expo}(D(v)) = e$, and all edges (v,w) with $\{\texttt{expo}(D(v)), \texttt{expo}(D(w))\} \subseteq \{e, e+1\}$. The latter includes all edges (v,w) with $\texttt{expo}(d(v)) = \texttt{expo}(d(w)) = e$. Subtract $e-\wp$ from all the exponents and convert to integers, rounding down, i.e. $x \mapsto \lfloor x/2^{e-\wp} \rfloor$. Call the integer USSSP algorithm, convert back to floats, and add $e-\wp$ to the exponents.

To see that the above is correct for any w with $\texttt{expo}(d(w)) = e$, consider a shortest path $v_0 \cdots v_l$ from $s = v_0$ to $w = v_l$. Let v_i be the last vertex with $\texttt{expo}(d(v_i)) < e$. By induction, $D(v_i) = d(v_i)$, and also, we contracted $v_i \in S$ correctly, setting $D(v_{i+1}) = D(v_i) \oplus \ell(v_i, v_{i+1}) = d(v_i) \oplus \ell(v_i, v_{i+1}) = d(v_{i+1})$. For $j = i+1, \ldots, l-1$, we have $\texttt{expo}(d(v_j)) = \texttt{expo}(d(v_{j+1})) = e$, so (v_j, v_{j+1}) is an edge in G_e. Moreover, for each of the desired floating point additions $D(v_j) \oplus \ell(v_j, v_{j+1})$, the first term and the sum both have exponent e. This implies that the integer additions performed by the integer USSSP algorithm simulate the desired floating point additions with exactly the right precision $\wp$. Thus, we end up with $D(w) = d(w)$, as desired.

Clearly, each edge is considered at most twice in its original form, and at most once as distance edge. Thus, the total time spent above is linear in the total number of edges.

Theorem 11 *There is a linear time Turing reduction from the floating point USSSP problem to the integer USSSP problem.*

Combining with the linear time algorithm for integer USSSP from [Tho97], we get

Corollary 12 *There is a linear time algorithm for USSSP problem with floating point weights.*

5 A remark on max-flow

Above it was shown how we can deal with floating points in connection with USSSP. In the introduction, we made some remarks concerning max-flow. The problem is that many max-flow algorithms include a factor $O(\log C)$ in their time bound, where C is the maximal capacity in the network. However, if the capacities are floating points, $\log C = \texttt{expo}(C)$, and $\texttt{expo}(C)$ may be exponentially large. We will address the problem briefly by showing that by at most doubling the error, we can restrict our attention to a network with maximal capacity C' where $\log C' = O(\log n + \wp)$.

Let f^* denote the (unknown) maximal flow value. First we find an s-t path with maximal minimal capacity D, that is, D is the maximal flow that can be pushed along a single path. Then $D \leq f^* \leq mD$. We can therefore reduce all capacities bigger than mD to mD without affecting the maximal flow. Concerning the small capacities, we can exploit that we are anyway going to accept an error in the order of $D/2^{\wp}$. At most doubling this error, we can take all weights and round down to the nearest multiple of $D/(m2^{\wp})$. More precisely, we set $e = \texttt{expo}(D) - \lfloor \log_2 m \rfloor - \wp$, subtract e from all exponents and round down to nearest integer. If C' is now the maximal capacity, $\log_2 C' \leq 2\lceil \log_2 m \rceil + \wp = O(\log n + \wp)$. Then we solve the max-flow problem with the reduced weights, and add e to the exponent of the resulting flow value.

Note above that we convert a floating point max-flow problem to one with bounded integer capacities. In practice, one may want to keep working with floats in order to benefit from an efficient floating point co-processor. However, working directly with floats in integer max-flow algorithms may, for example, cause infinite loops due to rounding problems. Such problems are addressed in [AM97].

Acknowledgement

I would particularly like to thank referee 4 from STACS'98 for some very thorough and useful comments.

References

[AH97] S. Albers and T. Hagerup, Improved parallel integer sorting without concurrent writing, *Information and Control* **136** (1997) 25–51.

[AM97] E. Althaus and K. Mehlhorn, Maximum Network Flow with Floating Point Arithmetic, Max-Plank-Institut für Informatik, Technical Report MPI-I-97-1-022, 1997.

[Dij59] E.W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.* **1** (1959), 269–271.

[FW94] M.L. Fredman and D.E. Willard, Trans-dichotomous algorithms for minimum spanning trees and shortest paths, *J. Comp. Syst. Sc.* **48** (1994) 533–551.

[KKT95] D.R. Karger, P.N. Klein, and R.E. Tarjan, A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees *J. ACM*, 42:321–328, 1995.

[Tho97] M. Thorup. Undirected Single Source Shortest Paths in Linear Time. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 12–21, 1997.

[vBKZ77] P. van Emde Boas, R. Kaas, and E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Syst. Th.* **10** (1977), 99–127.