Lecture notes (draft)
Single source shortest paths and Goldberg's scaling algorithm

László Kozma
Graphs & Algorithms, TU Eindhoven, Spring 2018

Consider a *directed*, *weighted* graph $G = (V, E, w)$, with vertex-set V, edge-set E, and weight-function $w(\cdot)$. The function $w(\cdot)$ assigns a real value to every edge (think of it as length or cost). We denote $n = |V|$ and $m = |E|$. Edges are ordered pairs of vertices, i.e. $E \subseteq V^2$. We say that there is an edge from u to v in G, if $(u, v) \in E$, where $u, v \in V$ and $u \neq v$.

A sequence of vertices $(x_1, x_2, \ldots, x_k)$ is a *path* from x_1 to x_k, if $(x_i, x_{i+1}) \in E$, for all $1 \leq i < k$; we say that these are the edges of the path. If there is a path from u to v, we say that v is *reachable* from u.

The length of a path $(x_1, x_2, \ldots, x_k)$ is the sum of the weights of its edges, i.e. $\sum_{i=1}^{k-1} w(x_i, x_{i+1})$. The distance from u to v, denoted $d(u, v)$ is the smallest length of a path from u to v. (Observe that the shortest path is not necessarily unique.) We are interested in the single-source shortest paths (SSSP) problem: finding the distances of all vertices from a specific (source) vertex $s \in V$, i.e. we want to find $d(s, x)$ for all $x \in V$. Typically we also want to find actual paths that realize these distances.

Another natural problem is the all-pairs shortest paths (APSP) problem which we do not discuss here.

A cycle is a sequence of edges $(x_1, x_2), (x_2, x_3), \ldots, (x_k, x_1)$. The length of a cycle is defined similarly to the length of a path.

Cycles with negative length are problematic. If a vertex on the negative cycle is reachable from u, and a vertex on the negative cycle can reach v, then the distance $d(u, v)$ is undefined. (By convention, we set it to $-\infty$.) We will therefore assume that the input graph contains no negative cycles. If v is not reachable from u, we define $d(u, v) = \infty$.

The topics of the first half are covered in most textbooks on algorithms, e.g. [1, 4]. The description of Goldberg's algorithm can be found in the original paper [3] and in lecture notes and slides, e.g. [5, 2, 6]. Our description here mostly follows [5].

In fact, most often we are interested in the SPSP (single-pair shortest path) problem. Unfortunately, apart from some special cases and heuristics, the best way we know to solve SPSP is by solving SSSP.

We could require vertices to appear at most once on a path, and then $d(s, \cdot)$ is defined even when there are negative cycles. However, in this case the problem becomes NP-hard: Suppose all edges are of weight -1. A path from s to x of length $-n + 1$ is a Hamiltonian path.

1 Non-negative weights

First we review the case when all weights are non-negative. The simplest is when all weights are equal to each other (we can assume that they are all 1). Then, the distance from u to v equals the number of edges needed to reach

v from u. In this case we can solve the SSSP problem using a BFS traversal of the graph from the source s.

Recall BFS.

Suppose now that the edge-weights are non-negative but not necessarily equal to each other. We don't make any further assumptions on the edge-weights, e.g. on their magnitudes – we just assume that we can store and retrieve them, and we can do elementary operations (comparison, addition) on them in constant time.

Recall that $s \in V$ is the source and we want to find the distances $d(s, x)$ for all $x \in V$. We already know $d(s, s) = 0$. Let us denote by $d[x]$ for every $x \in V$ the *tentative* distance from s to x, i.e. our current estimate of the distance. In the beginning, we set $d[s] = 0$ and $d[x] = \infty$ for all $x \in V \setminus \{s\}$. During the execution of an SSSP algorithm we update the values $d[x]$ and at the end of the algorithm $d[x]$ will store the correct value $d(s, x)$ for all x.

A fundamental operation that we frequently use is the **edge-relaxation**: $relax(u, v)$ sets $d[v]$ to $\min\{d[v], d[u] + w(u, v)\}$, i.e. it attempts to improve the distance to v via the edge (u, v).

Let us review **Dijkstra's algorithm** at a high level:

1. Initially set $W = V$. Let $d[s] = 0$, $d[x] = \infty$ for all $x \in V \setminus \{s\}$.
2. While W not empty, pick $x \in W$ that has minimum $d[x]$, and for all y such that $(x, y) \in E$ do $relax(x, y)$. Remove x from W.

Exercise 1: Extend the algorithm so that it also finds the actual shortest paths.

Why does this algorithm work? Observe the following invariants: (1) the tentative distance $d[x]$ of a vertex x is never smaller than the correct distance $d(s, x)$, and (2) when a vertex x is removed from W, its tentative distance is equal to its correct distance. (1) and (2) together imply the correctness of the algorithm.

Prove invariants (1)(2).

What is the running time of the algorithm? It depends on the data structures we use for implementing the operations. Observe that we need to maintain a set of size at most n, supporting the operations of extracting the minimum (n times) and decreasing the value of an item (m times, since every edge is relaxed at most once). Fibonacci heaps support the two operations at cost $O(\log n)$ and $O(1)$, respectively, which gives the overall running time $O(m + n \log n)$.

Exercise 2: Observe that the distances are returned in increasing order. Argue that for algorithms with this property we cannot hope for a better running time.

2 Negative weights

We now allow edges to be arbitrary reals, still disallowing negative cycles (in fact, the algorithms we look at can detect if a negative cycle is present). If negative weights are allowed, Dijkstra's algorithm breaks down.

Exercise 3: Construct an example with negative weights where Dijkstra's algorithm gives an incorrect answer.

We look at algorithms that can deal with negative weights.

Let us denote the edges in E as $e_1, e_2, \ldots, e_m$, in arbitrary order.

Exercise 4: Why can't we just fix negative weights by adding some large value to all weights?

The **Bellman-Ford** algorithm is very simple:

1. Let $d[s] = 0$, $d[x] = \infty$ for all $x \in V \setminus \{s\}$.
2. Repeat $n-1$ times: $relax(e_1)$, $relax(e_2)$, $\ldots$, $relax(e_m)$.

The running time is $O(mn)$ since we do $(n-1)m$ edge-relax operations, each taking constant time. This is much worse than the running time of Dijkstra's algorithm. Unfortunately, in general, this is still the best algorithm we know (after 60+ years).

Again, argue that we can also find the actual shortest paths with the same asymptotic running time.

For correctness, see that if the shortest path from s to x consists of the edges (in this order) $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$, then relaxing these edges (in this order) results in setting $d[x] = d(s, x)$. Relaxing additional edges cannot change the correctness of $d[x]$ (by the same argument as for Dijkstra).

Exercise 5: What happens if there is a negative cycle? How would you extend the Bellman-Ford algorithm to detect it?

Finally, observe that the sequence that consists of $(e_1, e_2, \ldots, e_m)$ repeated $n-1$ times, contains every possible edge-sequence of length at most $n-1$ as a subsequence (not necessarily contiguous). Furthermore, no shortest path can contain more than $n-1$ edges.

How would you try to speed up this algorithm in practice? When do we know that an edge-relax operation is unnecessary?

Vertex potentials. Consider a function $p : V \to \mathbb{R}$. We can change the edge weights for every $x, y \in V$ according to $p(\cdot)$ as follows:

$$w'(x, y) = w(x, y) + p(x) - p(y).$$

In words, we add to the old weight the potential-difference between the endpoints of the edges. We denote by $d'(u, v)$ the distance between u and v according to the modified weights w'.

The following properties are easy to prove with telescoping sums:

Do it!

(1) For every $x, y \in V$, we have $d'(x, y) = p(x) - p(y) + d(x, y)$.

(2) The shortest paths remain the same after the transformation.

(3) The length of every cycle remains the same after the transformation.

The potential function $p(\cdot)$ is called *feasible*, if the new weights are all non-negative, i.e. $w'(x, y) \geq 0$, for all $x, y \in V$. Observe that this means that $p(y) - p(x) \leq w(x, y)$, for every $x, y \in V$.

Such a potential function would be very useful: We could use it to make all weights non-negative (in $O(m)$ time), solve SSSP using Dijkstra, and transform the obtained distances back (in $O(n)$ time).

Unfortunately we don't know how to find a feasible potential function faster than solving the original problem. In fact, the two problems are essentially equivalent. If we have a negative cycle, then there clearly cannot be a feasible potential function. (Cycle lengths remain unchanged, so a negative cycle would contradict that all weights are non-negative.) On the other hand, if there are no negative cycles, we can always find a feasible potential function (by solving SSSP – see exercise).

The potential function trick is useful for solving APSP in the presence of negative weights: solve a single SSSP problem, get rid of the negative weights by using the distances as potential, run Dijkstra from every vertex in turn, transform back the distances. (This is known as Johnson's algorithm.)

Exercise 6: Show that for an *arbitrary* s from which every node is reachable, $p(x) = d(s, x)$ is a feasible potential function. If there is no such s, can you still find a feasible potential function? (Assuming there are no negative cycles.)

3 Goldberg's algorithm

In this section we look at Goldberg's algorithm for SSSP. Now we make a stronger assumption on the input than before: we require that the weights are *integers* in the range $[-N, \infty)$, for some N.

The running time of Goldberg's algorithm is then $O(m\sqrt{n}\log N)$, which is better than Bellman-Ford if $N = o(2^{\sqrt{n}})$.

3.1 The scaling framework

The algorithm is based on a technique known as *scaling*. This means that we first solve a rough approximation of our original problem, then we bring in more details, refining the solution, until, in the end, we have solved the original problem.

We only focus on finding a feasible potential function, or detecting the existence of a negative cycle (based on the earlier discussion, combined with Dijkstra's algorithm, this is sufficient to obtain the required running time).

Let us assume that we have as a black box an algorithm that finds, with running time $O(m\sqrt{n})$, a feasible potential function, if the weights are integers in the range $[-1, \infty)$. Call this algorithm *Refine*. (We will look into this black box in the next subsection.)

We proceed as follows: Recall that the weights are from $[-N, \infty)$. Let $L \geq 1$ be the smallest integer such that $N \leq 2^L$. We will do $L + 1$ iterations, with one call to *Refine* in each iteration. The claim on the total running time follows immediately.

In iteration i (for $i = L, \ldots, 0$), we define the weights $w^i(e) = \lceil \frac{w(e)}{2^i} \rceil$, for all $e \in E$. (This is our rough approximation of the problem.) We find a potential function $p^i(\cdot)$ that is feasible with weights $w^i(\cdot)$.

Iteration L (the first one) is easy, since all weights are positive (in fact, there's nothing to do in this iteration, $p^L(\cdot) = 0$ is feasible). Check!

In iteration 0 (the last one), we are back to the original weights, so if we got this far, it means that we have solved the original problem.

We just need to show how the intermediate step can be done.

Suppose that we have found $p^{i+1}(\cdot)$ feasible with $w^{i+1}(\cdot)$. This means that

$$w^{i+1}(x,y) + p^{i+1}(x) - p^{i+1}(y) \geq 0,$$

for all $x, y \in V$. Then

$$w^{i}(x,y) + 2p^{i+1}(x) - 2p^{i+1}(y) \geq -1.$$

(Recall definition of w^i.)

We call *Refine* for the problem with weights $w'(x,y) = w^i(x,y) + 2p^{i+1}(x) - 2p^{i+1}(y)$. We obtain the feasible potential function $p'(\cdot)$ for this problem.

Since

$$w'(x,y) + p'(x) - p'(y) \geq 0,$$

it follows that

$$w^{i}(x,y) + 2p^{i+1}(x) - 2p^{i+1}(y) + p'(x) - p'(y) \geq 0,$$

for all $x, y \in V$. Thus, $2p^{i+1}(\cdot) + p'(\cdot)$ is a feasible potential function with $w^i(\cdot)$, and this will be our $p^i(\cdot)$.

We claim that if G has a negative cycle, then one of the calls to *Refine* will fail (detecting a negative cycle in a subproblem). On the other hand, if there are no negative cycles in G, all the subproblems are feasible (no spurious negative cycle will be created).

Prove both claims!

3.2 The Refine step of Goldberg's algorithm

Recall that we want to find a feasible potential function $p(\cdot)$ for a graph G with integer edge weights from $[-1, \infty)$, in time $O(m\sqrt{n})$.

We make use of two classical results.

(1) A set of vertices $V' \subseteq V$ forms a *strongly connected component*, if for all $x, y \in V'$ there is a path from x to y. We can partition V into maximal strongly connected components in a unique way, in time $O(m)$.

(2) If the graph G has non-negative integer edge weights, and the largest distance is at most D then we can solve SSSP in $O(m + D)$ time. This is known as Dial's implementation of Dijktra's algorithm. (Hint: use an array instead of Fibonacci heap.)

Recall these facts.

Now look at G, and consider the graph G_- that is defined over the same vertices as G, but only with those edges of G that have weight -1 or 0. We use G_- to guide the construction of a potential function that is feasible for G. Initially we set $p(\cdot) = 0$ for all vertices of G.

First, we decompose G_- into strongly connected components. If there is an edge of weight -1 within any of the components, then we have a negative cycle (we can extend the edge to a cycle by using only non-positive edges within the component). In this case, we report the negative cycle and stop.

Otherwise, we *contract* the strongly connected components into super-vertices and treat them together. (Whenever we need to change the potential of a super-vertex to some value, we will in fact change the potentials of all its component-vertices to the same value.) This requires some careful data structuring, which we ignore for now.

We have more pressing things to worry about.

Observe that the graph we obtain fron G_- after contracting its components is acyclic. (In fact, it is a multigraph, since between components there may have been multiple edges and we keep all of them.)

From now on we assume that G_- is an acyclic multigraph. We call vertices that have an incoming -1-weight edge, *negative* vertices. Our task is to update the potential function $p(\cdot)$ so that we "fix" all negative vertices (so that they are no longer negative after we transform the weights using the potential function).

Let x be a negative vertex. If we simply reduce $p(x)$ by 1, then x is fixed,

since all incoming edges will increase by 1 after the transformation. However, edges outgoing from x will decrease their weight, possibly creating new negative vertices (or even decreasing some edge weights below -1). If, however we reduce by 1 the potential of x and at the same time of *all* vertices reachable from x in G_-, then we have fixed x without ruining any other vertex. (We use here that G_- is acyclic.) Observe that edges not in G_- may also decrease by 1, but these edges were at least 1 to begin with, so we don't create new negative edges.

The problem is that finding all vertices reachable from x may take $O(m)$ time, which is too much to pay for fixing a single vertex. We must, therefore try to fix several vertices at once.

Decomposing into levels. We add a virtual vertex s_0 to G_-, and connect it to all vertices, using 0-weight edges. We mark vertices of G_- according to their distance from s_0. Observe that these distances can be from $\{0, -1, \cdots, -n+1\}$. We refer to these values as the *label* of a vertex. The task of finding these values amounts to solving a SSSP problem. Luckily, the graph we are working with is acyclic.

Exercise 7: Show that in an acyclic graph (a DAG), we can solve SSSP in $O(m+n)$ time with arbitrary weights.

Suppose there are k negative vertices in G_-. We claim that either there is a group of at least $\lfloor\sqrt{k}\rfloor$ of them, such that they all have the same label, or otherwise, there is a shortest path from s_0 to some vertex, containing at least $\lfloor\sqrt{k}\rfloor$ negative vertices with different labels.

Exercise 8: Show this. The argument can be seen as a simpler form of Dilworth's theorem.

We show that in both cases we can make significant progress: we can at once fix at least $\lfloor\sqrt{k}\rfloor$ negative vertices, without creating any new ones or decreasing any edge-weights below -1. Before we show this, let's see how it would help.

Suppose we have k negative vertices (k could be as large as n). All the described steps (forming G_-, finding and contracting strongly connected components, adding virtual source, solving SSSP, finding large group with same labels or long path with different labels) and even the step we haven't yet described (fixing the $\lfloor\sqrt{k}\rfloor$ negative vertices) can be done in $O(m+n)$ time. We have thus reduced the number of negative vertices by $\lfloor\sqrt{k}\rfloor$. The number of iterations is at most $T(n)$, where $T(k) \leq 1 + T(k - \lfloor\sqrt{k}\rfloor)$. This resolves to $T(n) = O(\sqrt{n})$. Multiplying by the cost of an iteration, we get the required $O(m\sqrt{n})$.

Think it through. Are all data structures obvious?

Prove it!

Now it's time for the actual fixing.

Fixing a level. Suppose we have at least $\lfloor\sqrt{k}\rfloor$ vertices all with the same label $-t$, i.e. all at the same distance $-t$ from the virtual source s_0. Then decrease by 1 the potential of all vertices with labels $t' \leq -t$. This fixes all vertices with label $-t$, without decreasing any edge weight to a negative value. To see this, suppose for contradiction that $w(x, y)$ is decreased to -1 or below. Then $w(x, y) \leq 0$ must have been true before the operation, therefore (x, y) was an edge in G_-. But $w(x, y)$ can decrease only if we had $d(s_0, x) > -t$, and $d(s_0, y) \leq -t$. This contradicts that $d(s_0, \cdot)$ is valid, because a $relax(x, y)$ operation would have decreased the distance from s_0 to y via x.

Fixing a path. In this case we have to work a bit harder.

It may help to sketch an illustration as we go.

Suppose we have a shortest path in which there are at least $\lfloor\sqrt{k}\rfloor$ vertices with different labels. Then there must be at least $\lfloor\sqrt{k}\rfloor$ negative vertices that have an incoming negative edge on the path. Denote these vertices as $w_1, w_2, \ldots, w_r$.

Consider a graph G_+ with the same vertices and edges as G, but with the weights changed as follows: the weight of an edge e in G_+ is $w(e)$ if $w(e) \geq 0$, and 0 otherwise. (We change all negative weights to 0.)

Add a virtual source s_{00} to G_+ and add edges connecting s_{00} to every other vertex. The weights of these edges are as follows: the weight of (s_{00}, w_i) is $r - i$, for all i. The weight of every other newly added edge is r.

Compute SSSP in this augmented G_+, and let the potential function be $p(x) = d(s_{00}, x)$ for all x. Observe that the weights are non-negative integers, in the range $[0, n]$. We claim that in this case we can compute SSSP in $O(m)$ time (using Dial's implementation of Dijkstra – observe that all distances are smaller than n).

It remains to verify that the potential function we obtained (1) fixes $w_1, \ldots, w_r$ in G_- (unless there is a negative cycle), (2) doesn't create new negative weight edges, and (3) doesn't make an edge-weight drop below -1.

Let's do it:

(1) Look at a negative edge (x, w_i) going to some vertex w_i in G_-. If we obtained $d(s_{00}, x) > d(s_{00}, w_i)$, then the negative edge was fixed. Suppose the opposite ($d(s_{00}, x) \leq d(s_{00}, w_i)$). Observe that $d(s_{00}, w_i) \leq r - i$. If $d(s_{00}, x) \leq r - i$, the first vertex after s_{00} on the shortest path to x must

have been w_i or some w_j with $j \geq i$. (Otherwise the first edge would have been already longer than $r - i$.) Then $d(s_{00}, x) = r - j + d(w_j, x) \leq r - i$, from which it follows that $d(w_j, x) \leq j - i$.

But then, in the original graph we must have the following negative cycle: w_i to w_j (length not more than $i - j$), w_j to x (length not more than $j - i$), edge (x, w_i) (weight -1).

(2) Suppose some edge (x, y) becomes negative, i.e. earlier it was $w(x, y) \geq 0$. But then the weight of this edge in G_+ remained unchanged. It must hold that $d(s_{00}, y) \leq d(s_{00}, x) + w(x, y)$ (by triangle-inequality), which contradicts that the new edge weight $w(x, y) + p(x) - p(y)$ is negative.

(3) Suppose $w(x, y) = -1$ and the weight decreases even more. Observe that the weight of this edge in G_+ was set to 0. It must hold that $d(s_{00}, y) \leq d(s_{00}, x)$ (by triangle-inequality), which contradicts that $w(x, y) + p(x) - p(y) < -1$.

We are done.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[2] Andrew V. Goldberg. Slides. `http://www.avglab.com/andrew/pub/path05-slides.pdf`.

[3] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[4] Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, USA, 1983.

[5] Uri Zwick. Lecture notes. `http://www.cs.tau.ac.il/~zwick/grad-algo-09/short-path.pdf`.

[6] Uri Zwick. Slides. `http://www.cs.tau.ac.il/~zwick/grad-algo-17/SP-Negative-Cycles.pptx`.

Exercises

- Exercises 1–8 on the margin
- The *width* of a path is defined as the minimum weight of an edge on the path. We would like to find maximum width paths from a source vertex to every other vertex. Adapt Dijkstra to achieve this, with the same running time.
- You are given the pairwise exchange rates between various currencies. How would you find an opportunity for arbitrage? (A sequence of exchanges guaranteed to yield a profit.)
- Given is a graph with edges colored red and blue. (1) Report whether there is a path from u to v with more red edges than blue edges. (2) Find the path from u to v with the smallest number of blue edges. (3) Find if there is a path from u to v with at least 3 red edges. (4) Find if there is a path from u to v with exactly 3 red edges. What is the running time of your solutions?
- Consider the following variant of Dijkstra's algorithm, which we could call *insistent-Dijkstra*. Show that the algorithm correctly solves the SSSP problem even if there are negative weights (but no negative cycles). Show that the number of steps may be exponential.

  Let $G = (V, E, w)$ be a weighted, directed graph.

  1. Initially set $W = V$. Let $d[s] = 0$, $d[x] = \infty$ for all $x \in V \setminus \{s\}$.
  2. While W not empty, *pop* x from W, and for all y such that $(x, y) \in E$ do *relax*(x, y). If $d[y]$ has changed, *push* y to W.

  We did not specify the implementation of W here. Suppose first that W is a priority queue as in the standard Dijkstra: *pop* returns the smallest item. Alternatively, consider the case when W is implemented as a *stack* (last in – first out), or as a *queue* (first in – first out). The initial ordering of the vertices in W is arbitrary.

  Is there any connection/similarity between *insistent-Dijkstra* and Bellman-Ford?