Lecture 8

Distance Approximation and Routing

Knowing how to construct a minimum spanning tree is very useful to many problems, but it is not always enough. Cheaply connecting all nodes is one thing, but what about finding a short path between an arbitrary pair of nodes? Clearly, an MST is not up to the task, as even for a single edge, the route in the MST might be factor $n-1$ more costly: just think of a cycle!

Trivially, finding the distance between two nodes is a global problem, just like MST. However, the connection runs deeper. As we saw in the exercises for the previous lecture, even just *approximating the weight* of a shortest *s–t* path requires $\Omega(\sqrt{n}/\log^2 n + D)$ rounds in the worst case (with messages of size $\mathcal{O}(\log n)$).

Doing this every time a routing request is made would take too long and cause a lot of work. We're too lazy for that! So instead, we will preprocess the graph and construct a distributed data structure that will help us serving such requests.

Definition 8.1 (Distributed all-pairs-shortest-path problem (APSP)). *In the* all-pairs-shortest-path *problem, we are given a weighted, simple, connected graph $G = (V, E, W)$. The task is for each node $v \in V$ to compute a* routing table, *so that given an identifier $w \in V$, v can determine $d(v, w)$, the (weighted) distance from v to w, i.e., the minimum weight of a path from v to w, and the next node on a shortest path from v to w. For $\alpha > 1$, an α-approximation merely guarantees the that the stated distance d satisfies $d(v, w) \leq d \leq \alpha d(v, w)$ and that routing path has weight at most $\alpha d(v, w)$.*

We will solve this problem in the synchronous message passing model without faults. In other words, we will accept some preprocessing out of necessity, but refuse to fall back to a centralized algorithm!

APSP is hard

As mentioned above, we know that we should not even try to find an algorithm faster than (roughly) $\Omega(\sqrt{n})$.

Corollary 8.2. *An α-approximate solution to APSP with $\mathcal{O}(\log n)$-bit messages requires $\Omega(\sqrt{n}/\log^2 n + D)$ rounds, regardless of α.*

However, things are much worse. Even in an unweighted (i.e., $W(e) = 1$ for all $e \in E$) *tree of depth* 2, solving APSP requires $\Omega(n/\log n)$ rounds!

Theorem 8.3. *Any deterministic α-approximation to APSP with $\mathcal{O}(\log n)$-bit messages requires $\Omega(n/\log n)$ rounds, even in trees of depth 2.*

Proof. Consider a tree whose root has two children, which together in total have k children with identifiers $1, \ldots, k$. We consider all such graphs. Note that the root can only tell them apart by the bits that its two children send, which are $\mathcal{O}(R \log n)$ for an R-round algorithm. The number of different routing tables the root may produce is thus $2^{\mathcal{O}(R \log n)}$. Note also that for each of the considered graphs, we need a different routing table: any two partitions of k nodes must differ in at least one node, for which the routing table then must output a different routing decision. How many such partitions are there? Well, 2^k – just decide for each node $1, \ldots, k$ to which child of the root it's attached. Hence,

$$k \in \mathcal{O}(R \log n),$$

or $R \in \Omega(k/\log n) = \Omega(n/\log n)$, as the considered graph family has $n = k + 3$ nodes. □

Will randomization save us? Not today. If the number of bits received by the root is too small, it will in fact err with a large probability.

Corollary 8.4. *Any randomized α-approximation to APSP with $\mathcal{O}(\log n)$-bit messages requires $\Omega(n/\log n)$ rounds, even in trees of depth 2.*

Proof. Suppose there's a randomized algorithm that terminates in $o(n/\log n)$ rounds. Fix the random bit strings of all nodes, and execute the resulting *deterministic* algorithm, on a *uniformly random* topology as in the proof of the theorem. Now, as the there are $2^{o(n)}$ different bit strings the root can possibly receive in $R \in o(n/\log n)$ rounds, the probability that the algorithm computed a correct table is at most $2^{o(n)}/2^n = 2^{(1-o(1))n}$, i.e., astronomically small. As we used the same random distribution of topologies *irrespectively of the assigned random bits*, we can now argue that choosing the random bit strings of the nodes uniformly and independently at random *after* we picked the topology yields the same result. □

Remarks:

- The above corollary is an application of *Yao's principle.* If one provides a distribution of inputs, no randomized algorithms can perform better than the best deterministic algorithm for this distribution.
- For exact algorithms with weights, the bound becomes $\Omega(n)$: just add a weight from $1, \ldots, n$ to each edge to a leaf, resulting in n^k distinct combinations, even with a single child!
- This also shows that if we only care about distances (and not how to route), we're still skrewed. Even just approximate distances: For any reasonable approximation, we can make sure that there are at least two different "classes" of distances for each node that need to be distinguished.

- Essentially, the bound still holds even if we permit *dynamic* routing (without knowing distances), where nodes on the routing path may attach some routing information to the message. This way, one can "check" whether the destination is attached to a child and return to the root if the decision was wrong. One then uses $\Theta(\rho)$ children of the root to show that a ρ-approximation (even on average) is not possible in $o(n/(\rho^2 \log n))$ rounds.
- In n rounds, everyone can learn the entire tree, so at least for this family of graphs the bound is tight. Let's see what we can do for arbitrary graphs!

Exact APSP in unweighted graphs

If a problem appears to be difficult, one shouldn't always try to take on the most general form first. We start by considering unweighted graphs.[1] In a nutshell, solving APSP here is equivalent to constructing for each node a BFS tree rooted at it.

The setting is synchronous, so we know how to do this for a single node in $\mathcal{O}(D)$ rounds. The challenge is that the different constructions might interfere. We have seen that we cannot avoid this completely, as it will take $\Omega(n)$ rounds even if $D \in \mathcal{O}(1)$, but we can still hope for a running time that is much faster than the trivial solution of running n instances of the Bellman-Ford algorithm sequentially, i.e., $\Theta(Dn)$ rounds.

It turns out that there is a straightforward solution to this problem.[2] We employ Bellman-Ford for all sources concurrently, where always the seemingly most useful piece of information is communicated. "Seemingly most useful" here means to always announce the closest node that hasn't been announced before, breaking ties by identifiers. "Source" refers to a node $s \in S \subseteq V$; as we will see, the algorithm works very well for the more general setting where only distances to a subset $S \subseteq V$ of nodes are to be determined.

Definition 8.5 (Total order of distance/node pairs). *Let $(d_v, v), (d_w, w) \in \mathbb{N}_0 \times V$ be two distance/node pairs. Then*

$$(d_v, v) < (d_w, w) \Leftrightarrow (d_v < d_w) \vee (d_v = d_w \wedge v < w).$$

Here the comparison "$v < w$" means to numerically compare the identifiers of v and w.

In the following we will consider all sets of distance/node pairs to be ordered ascendingly according to the above definition (and consequently refer to them as lists).

Let's fix some helpful notation.

Definition 8.6. *For each node $v \in V$ and each round $r \in \mathbb{N}$, denote by L_v^r the content of v's L_v variable at the end of round r; by L_v^0 we denote the value at initialization. Furthermore, define $L_v := \{(d(v,s), s) \mid s \in S\}$ (this is slight abuse of notation; we will show that the algorithm returns exactly this L_v, though). For $h \in \mathbb{N}_0$, denote by $L_v(h)$ the sublist of L_v containing only elements $(d(v,s), s)$ with $d(v,s) \leq h$. For $k \in \mathbb{N}$ denote by $L_v(h,k)$ the sublist of the (up to) k first elements of $L_v(h)$.*

[1] That's not how we did it, but there's no reason you shouldn't learn from our mistakes!
[2] Actually several, but we're going for the one that will be most useful later on.

Algorithm 17 Pipelined Bellman-Ford, code at node v. Initially, v knows whether it is in S, as well as parameters $H, K \in \mathbb{N}$. Remembering the sender for each entry in L_v reveals the next routing hop on a shortest path to the respective source w.

```
if $v \in S$ then
    $L_v := \{(0, v)\}$
else
    $L_v := \{\}$
end if
for $i = 1, \dots, H + K - 1$ do
    $(d_s, s) :=$ smallest element of $L_v$ not sent before ($\perp$ if there is none)
    if $(d_s, s) \neq \perp$ then
        send $(d_s + 1, s)$ to all neighbors
    end if
    for each $(d_s, s)$ received from a neighbor do
        if $\nexists (d'_s, s) \in L_v : d'_s \leq d_s$ then
            $L_v := L_v \cup \{(d_s, s)\}$
        end if
        if $\exists (d'_s, s) \in L_v : d'_s > d_s$ then
            $L_v := L_v \setminus \{(d'_s, s)\}$
        end if
    end for
end for
return $L_v$
```

We will show that Algorithm 17 guarantees that after r rounds, for $h + k \leq r + 1$, the first $|L_v(h, k)|$ entries of L_v^r are already correct. Inserting $h = D$ and $k = n$, we will then see that the algorithm indeed returns the lists L_v.

With the right induction hypothesis, the proof is actually going to be quite simple. Let's assemble the pieces first.

Lemma 8.7. *If $(d_w, w) \in L_v^r$ for any $r \in \mathbb{N}_0$, then $w \in S$ and $d_w \geq d(v, w)$.*

Proof. We never add entries for nodes that are not in S. Moreover, initially for each $s \in S$ only s has an entry $(0, s) \in L_s^0$. As we increase the d-values by one for each hop, it follows that $d_s \geq d(v, s)$ for any entry $(d_s, s) \in L_v^r$. □

Corollary 8.8. *If for any $s \in S$ and $v \in V$, it holds that v receives $(d(v, s), s)$ from a neighbor in round $r \in \mathbb{N}$ (or already stores it on initialization), then $(d(v, s), s) \in L_v^{r'}$ for all $r' \geq r$. Moreover, if $L_v(h, k) \subseteq L_v^r$ for any $r \in \mathbb{N}_0$, it is in fact the head of the list L_v^r.*

Lemma 8.9. *For all $h, k \in \mathbb{N}$ and all $v \in V$,*

$$L_v(h, k) \subseteq \{(d(w, s) + 1, s) \,|\, (d(w, s), s) \in L_w(h - 1, k) \wedge \{v, w\} \in E\}\,.$$

Proof. For each $(d(v, s), s) \in L_v(h, k)$, consider a neighbor w of v on a shortest path from v to s. We have that $d(w, s) = d(v, s) - 1 \leq h - 1$. Hence, it suffices to show that $(d(w, s), s) \in L_w(h - 1, k)$. Assuming otherwise, there are k elements $(d(w, s'), s') \in L_w(h - 1, k)$ satisfying that $(d(w, s'), s') \leq (d(w, s), s)$. Hence, $(d(v, s'), s') \leq (d(w, s') + 1, s') \leq (h, s')$, and if $d(v, s') = d(v, s)$, then also

$d(w,s') = d(w,s)$ and thus $s' < s$. It follows that $(d(v,s'),s') < (d(v,s),s)$. But this means there are at least k elements in $L_v(h,k)$ that are smaller than $(d(v,s),s)$, contradicting the definition of $L_v(h,k)$! $\square$

Now we can prove the statement sketched above.

Lemma 8.10. *For every node $v \in V$, $r \in \{0,\dots,H+K-1\}$, and $h+k \leq r+1$,*

(i) $L_v(h,k) \subseteq L_v^r$, and

(ii) v has sent $L_v(h,k)$ by the end of round $r+1$.

Proof. We show the statement by induction on r. It trivially holds for $h = 0$ and all k, as $L_v(0,k) = \{(0,v)\}$ if $v \in S$ and $L_v(0,k) = \emptyset$ otherwise, and clearly this will be sent by the end of round 1. In particular, the claim holds for $r = 0$.

Now suppose both statements holds for $r \in \mathbb{N}_0$ and consider $r+1$. As the case $h = 0$ is already covered, we may assume that $h > 0$. By the induction hypothesis (Statement (ii) for r), for $h+k \leq r+1$, node v has already received the lists $L_w(h-1,k+1)$ and $L_w(h,k)$ from all neighbors w. By Lemma 8.9, v thus has received all elements of $L_v(h,k+1)$ and $L_v(h+1,k)$. By Corollary 8.8, this implies Statement (i) for $h+k \leq r+2$.

It remains to show Statement (ii) for $h+k = r+2$. Since we just have shown (i) for $h+k = r+2$, we know that $L_v(h,k) \subseteq L_v^{r+1}$ for all $h+k = r+2$. By Corollary 8.8, these are actually the first elements of L_v^{r+1}, so v will sent the next unsent entry in round $r+2$ (if there is one). By the induction hypothesis, v sent $L_v(h,k-1)$ during the first $r+1$ rounds (where $L_v(h,0) := \emptyset$), hence only $L_v(h,k) \setminus L_v(h,k-1)$ may still be missing. As $|L_v(h,k) \setminus L_v(h,k-1)| \leq 1$ by definition, this proves (ii) for $h+k = r+2$. This completes the induction step and thus the proof. $\square$

Corollary 8.11. *APSP on unweighted graphs can be solved with message size $\mathcal{O}(\log n)$ in $n + \mathcal{O}(D)$ rounds.*

Proof. We construct a BFS tree, count the number of nodes and determine the depth d of the BFS tree; this takes $\mathcal{O}(D)$ rounds, and we have that $d \leq D \leq 2d$. The root then initiates Algorithm 17 with $S = V$, $H = 2d$, and $K = n$, so that all nodes jointly start executing it in some round $R_0 \in \mathcal{O}(D)$. As for $S = V$, $L_v = L_v(D,n) = L_v(2d,n)$ (and remembering senders yields routing information), Lemma 8.10 shows that this solves APSP. $\square$

Remarks:

- Somewhat depressing, but we have seen that this is essentially optimal.
- We've actually shown something stronger. For *any* $S \subseteq V$ and *any* $h,k \in \mathbb{N}$, we can determine $L_v(h,k)$ at all nodes $v \in V$ in $h+k-1$ rounds.
- There's a straightforward example showing that this is the best that's possible for *any* h and k. Even more depressing!
- What do we do when we're depressed by lower bounds? We change the rules of the game!

Relabeling

Basically, the lower bound might mean that we haven't asked the right question. The problem is that we insisted on using the original identifiers. If there are bottleneck edges – like in the above construction the edges between the root and its children – this dooms us (modulo nitpicking over details) to transmit them all over these edges. The problem is easily resolved if we permit *relabeling*.

Definition 8.12 (APSP with Relabeling)**.** *The APSP problem with* relabeling *is identical to the APSP problem, except that each node now also outputs a* label*. The task is now to construct a routing table and a label $\lambda(v)$ at each node v so that, given $\lambda(w)$ of some node w, v can determine the distance and next routing hop to w. Approximate solutions are defined as before.*

How does this help us? Let's consider a peculiar special case first: in a tree, we want to be able to route from the root to each node.

Lemma 8.13. *Suppose we are given tree (V, E) of depth d. Using messages of size $\mathcal{O}(\log n)$, in $\mathcal{O}(d)$ rounds we can determine routing tables and assign labels $1, \ldots, |V|$ such that given the label $\lambda(v)$ of node $v \in V$, we can route from the root to v.*

Proof. We enumerate the tree nodes in a depth-first-search manner, i.e., according to the order in which they are visited by an Euler tour of the tree. In a distributed fashion, this is done as follows.

1. Determine for each $v \in V$ the number of nodes in its subtree. This is done in a bottom-up fashion in $\mathcal{O}(d)$ rounds: each node announces the number of nodes in its subtree to its parent, starting from the leaves.

2. The root labels itself 1 and assigns to each child a range of labels matching the size of its subtree. Each child then takes the first label from its assigned range and splits the remaining labels between its children in the same way. This top-down procedure takes $\mathcal{O}(d)$ rounds, too. Note that since the assigned ranges are consecutive, they can be communicated using $\mathcal{O}(\log n)$ bits by announcing the smallest and largest element of the respective interval.

The tables at each node store the label ranges assigned to the children. Hence, given a label $\lambda(w)$ of a node w, each node on the unique path from the root to w can determine the next routing hop. □

Remarks:

- This construction is inefficient in terms of memory, i.e., the size of tables. In a tree, one can be much more efficient and have tables of size $\log^{\mathcal{O}(1)} n$, without increasing label size significantly.

- We can also make distances available. Each node learns its distance to the root ($\mathcal{O}(d)$ rounds; simply do a "flooding" that sums up the weights of traversed edges) and adds it to the label. The resulting labels have size $\mathcal{O}(\log n)$.

- While handling trees does not seem very impressive, the labels help circumvent the bottleneck problem we might experience with the original identifiers. Let's now handle general (unweighted) graphs!

Fast APSP with relabeling: the unweighted case

From a previous exercise, we know that approximating the diameter of an unweighted graph better than factor 3/2 takes $\Omega(n/\log n)$ rounds. Hence, we'll have to live with getting only an approximation if we want to obtain a faster algorithm. The key idea in Algorithm 18 is to use a small set of "landmarks" to navigate to distant nodes, while handling close-by nodes directly.

The algorithm is for label and table construction. Before we discuss that it can be implemented quickly, let's first explain how we can route and estimate distances with approximation factor at most 5. For routing and distance estimation, given a label $\lambda(w)$ at a node v, v does the following:

- If $\exists(d(v,w),w) \in L_v(\sqrt{n\log n}, \sqrt{n\log n})$ for sources V, then v knows $d(v,w)$ and knows the next hop on a shortest path to w.
- Otherwise, we first route from v to s_w (s_w is part of $\lambda(w)$) using the tables for $L_v(\tilde{D},|S|) = L_v(D,|S|)$ and then from s_w to w using the tree label $\lambda_{s_w}(w)$. The distance is estimated as $d(v,s_w)+d(s_w,w)$, where $d(s_w,w)$ is available from $\lambda_{s_w}(w)$.

Let's first show that this is indeed a factor-5 approximation if a certain condition is met.

Lemma 8.14. *Suppose $(d(v,s_v),s_v) \in L_v(\sqrt{n\log n}, \sqrt{n\log n})$ with source set V for all $v \in V$, then the above routing and distance approximation scheme has approximation factor at most* 5.

Proof. If $(d(v,w),w) \in L_v(\sqrt{n\log n}, \sqrt{n\log n})$, then the solution is optimal. If not, the assumption that $(d(v,s_v),s_v) \in L_v(\sqrt{n\log n}, \sqrt{n\log n})$ implies that $(d(v,s_v),s_v) \le (d(v,w),w)$. In particular, $d(v,s_v) \le d(v,w)$. As by definition $d(w,s_w) \le d(w,s_v)$, the triangle equality yields that

$$\begin{aligned} d(v,s_w)+d(s_w,w) &\le d(v,w)+d(w,s_w)+d(s_w,w)\\ &\le d(v,w)+2d(w,s_v)\\ &\le d(v,w)+2(d(w,v)+d(v,s_v))\\ &\le 5d(v,w). \end{aligned}$$

□

Algorithm 18 5-approximate APSP with relabeling in unweighted graphs. By c we denote a sufficiently large constant.

```
1: determine $n$ and $\tilde{D} \in [D, 2D]$ and make both known to all nodes
2: sample each node into $S \subseteq V$ with independent probability $c\sqrt{\log n/n}$
3: determine $|S|$ and make it known to all nodes
4: add to each nodes' identifier a bit indicating whether it is in $S$
5: for source set $S$, compute $L_v(\tilde{D},|S|)$ for all $v \in V$
6: for each $v \in V$, $s_v := \text{argmin}_{s\in S}\{d(v,s)\}$
7: for each $s \in S$, compute labels $\lambda_s(v)$ for routing/distances to from the root
   $s$ of the (partial) BFS tree with nodes $\{v \in V \mid s_v = s\}$ rooted at $s$
8: relabel each $v \in V$ by $\lambda(v) := (s_v, \lambda_{s_v}(v))$
9: for source set $V$, compute $L_v(\sqrt{n\log n}, \sqrt{n\log n})$
10: return labels $\lambda(v)$ and all computed tables
```

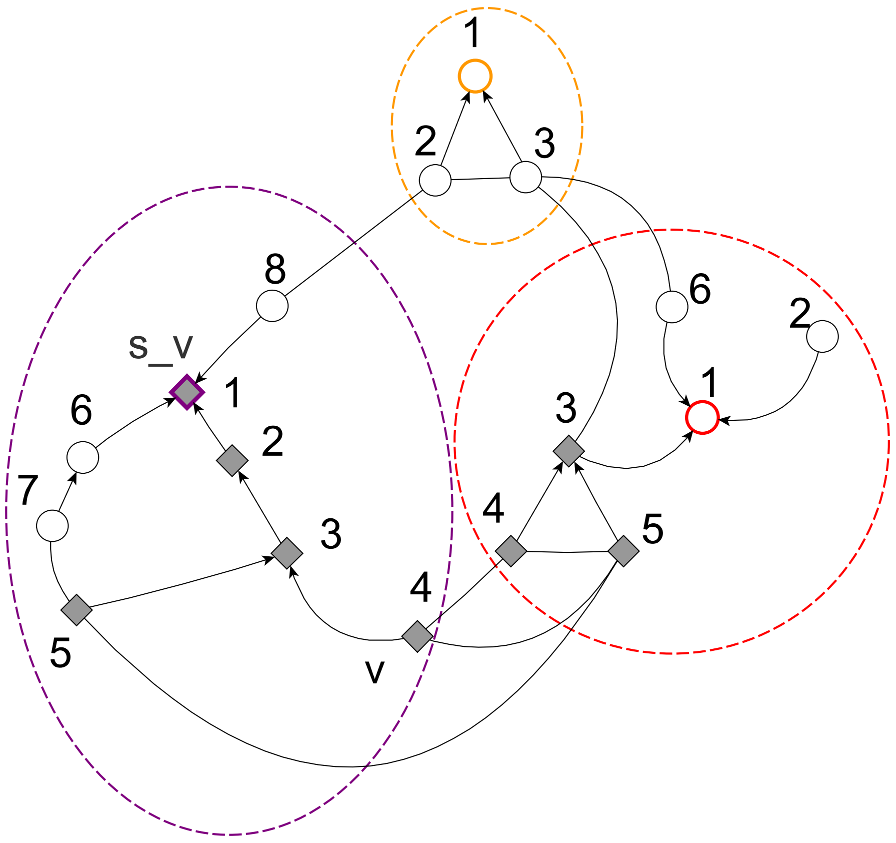

Figure 8.1: An example of the "clustering" constructed for the hierarchical routing scheme. The dotted ovals indicate the regions belonging to the sampled node framed in the same color. The oriented edges are part of the shortest-path tree rooted at that node. Each node is labeled by the identifier of its root and the number assigned to it according to the DFS enumeration of the trees (only these are written next to the nodes). The grey nodes are those for which node v (labeled $(s_v, 4)$) knows how to route directly to.

Using Chernoff's bound, it's straightfoward to see that the prerequisite of this lemma is satisfied w.h.p.

Lemma 8.15. *W.h.p., $(d(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for all $v \in V$.*

Proof. We sampled nodes into S with independent probability $c\sqrt{\log n/n}$. The expected number of nodes from S among a set of at least $\sqrt{n \log n}$ nodes – in particular the nodes indicated by $L_v(\sqrt{n \log n}, \sqrt{n \log n})$ for a given $v \in V$ – is thus at least $c \log n$. By Chernoff's bound, the probability that the number of such nodes is fewer than $c \log n/2$ is $2^{-\Omega(c \log n)} = n^{-\Omega(c)}$. As the constant c is assumed to be sufficiently large, we conclude that for each $v \in V$, it holds that $(d(v, s_v), s_v) \in L_v(\sqrt{n \log n}, \sqrt{n \log n})$ w.h.p. By the union bound, the joint event that this holds for all $v \in V$ occurs w.h.p., too. $\square$

It remains to understand the time complexity of the construction. An immediate consequence of the above lemma is that the partial BFS trees rooted at the nodes in S are not too deep.

Corollary 8.16. *W.h.p., the partial BFS trees rooted at the nodes $s \in S$ containing the nodes $\{v \in V \mid s_v = s\}$ all have depth $\mathcal{O}(\sqrt{n \log n})$.*

Now we just need to check the complexities of the individual steps.

Corollary 8.17. *Algorithm 18 can be implemented such that it terminates in $\mathcal{O}(\sqrt{n \log n} + D)$ rounds w.h.p.*

Proof. Lines 2, 4, 6, 8, and 10 are local computations only. Lines 1 and 3 can be done in $\mathcal{O}(D)$ rounds by constructing and using a BFS tree. By Lemma 8.10, calling Algorithm 17 with source set S, $H = \tilde{D} \in \Theta(D)$, and $K = |S|$ will handle Line 5. By Chernoff's bound, $|S| \in \Theta(\sqrt{n \log n})$ w.h.p., i.e., this takes $\mathcal{O}(\sqrt{n \log n} + D)$ rounds w.h.p. By Lemma 8.13 and Corollary 8.16, Line 7 can be completed in $\mathcal{O}(\sqrt{n \log n} + D)$ rounds. By Lemma 8.10, calling Algorithm 17 with source set V and $K = H = \sqrt{n \log n}$ will yield lists containing $L_v(\sqrt{n \log n}, \sqrt{n \log n})$; by Lemma 8.7, we can obtain the lists $L_v(\sqrt{n \log n}, \sqrt{n \log n})$ by discarding all entries (d_w, w) with $d_w > \sqrt{n \log n}$ and truncating the list to (at most) $\sqrt{n \log n}$ elements. Summing this all up, we get the claimed running time bound. □

Theorem 8.18. *In unweighted graphs, we can find a 5-approximate solution to APSP using messages of size $\mathcal{O}(\log n)$ in $\mathcal{O}(\sqrt{n \log n} + D)$ rounds w.h.p.*

Proof. By Lemmas 8.14 and 8.15, the approximation guarantee holds w.h.p. By Corollary 8.17, the time bound is satisfied w.h.p. □

Remarks:

- One can reduce the approximation factor to 3 if one permits access to routing tables of *both* source *and* destination when determining where to route/approximating the distance, as then one can route via s_v ro s_w, whatever is shorter.
- This is the typically done in centralized constructions, where the main point is to make the tables small. In this context it makes sense to be able to access both tables, but in the distributed setting this would defeat the purpose.
- The argument in Lemma 8.14 can be used repeatedly for a sampling hierarchy of k levels (i.e., each node makes it to the next level with probability roughly $n^{-1/k}$), resulting in an $\mathcal{O}(k)$-approximation. This yields a running time of $\mathcal{O}(k(n^{1/k}\sqrt{\log n} + D))$. An one can make the tables to have size about $n^{1/k}$, too!

Weighted APSP

In order to handle the weighted case, we'll reduce it to a small number of unweighted instances. Denote by $W_{\max} := \max_{e \in E}\{W(e)\}$ is the maximum edge weight. Fix any constant $0 < \varepsilon \leq 1$ and define for $i = \{0, \ldots, i_{\max}\}$, where $i_{\max} := \lceil \log_{1+\varepsilon} W_{\max} \rceil$, that $b_i := (1+\varepsilon)^i$ and $G_i := (V, E, W_i)$, where $W_i(e) := b_i \lceil W(e)/b_i \rceil$, i.e., G_i is G with edge weights rounded up to integer multiples of b_i.

Denoting by $d_i(v,w)$ the distance of v and w in G_i, obviously we have that $d_i(v,w) \geq d(v,w)$. The interesting bit is that there's a "sweet spot" for which $d_i(v,w) \leq (1+\varepsilon)d(v,w)$ and the $d_i(v,w)$ is a small multiple of b_i.

Lemma 8.19. *Denote by $h_{v,w}$ the hop length of a shortest path from v to w. For $i_{v,w} := \max\{0, \lfloor \log_{1+\varepsilon}(\varepsilon d(v,w)/h_{v,w}) \rfloor\}$, it holds that $d_{i_{v,w}}(v,w) \leq (1+\varepsilon)d(v,w) \in \mathcal{O}(b_{i_{v,w}} h_{v,w})$.*

Proof. If $i_{v,w} = 0$, we have that $d_{i_{v,w}} = d(v,w)$. Otherwise,

$$d_{i_{v,w}}(v,w) \leq d(v,w) + b_{i_{v,w}} h_{v,w} \leq (1+\varepsilon)d(v,w) \in \mathcal{O}(d(v,w))$$

Regarding the second inequality, observe that

$$d(v,w) = \frac{h_{v,w}}{\varepsilon} \cdot \frac{\varepsilon d(v,w)}{h_{v,w}} \leq \frac{h_{v,w}}{\varepsilon} \cdot (1+\varepsilon) b_{i_{v,w}} \in \mathcal{O}(b_{i_{v,w}} h_{v,w}). \qquad \square$$

Theorem 8.20. *For any constant $\varepsilon > 0$, we can $(1+\varepsilon)$-approximate APSP in $\mathcal{O}(n \log n)$ rounds with messages of size $\mathcal{O}(\log n)$.*

Proof. By Lemma 8.19, for all $v, w \in V$ we have that

$$d_{i_{v,w}}(v,w) \leq (1+\varepsilon)d(v,w) \in \mathcal{O}(b_{i_{v,w}} h_{v,w}).$$

Replace for each G_i each edge of weight kb_i by a virtual path of k edges of weight 1. The result is an unweighted graph $\tilde{G}_i$. Denoting by $L_{i,v}(h,k)$ the list for graph $\tilde{G}_i$, the lemma thus states that if we determine $L_{i_{v,w},v}(\mathcal{O}(h_{v,w}), n) = L_{i_{v,w},v}(\mathcal{O}(n), n)$, then there is an entry $(d, w) \in L_{i_{v,w},v}(\mathcal{O}(n), n)$ such that $b_{i_{v,w}} d \leq (1+\varepsilon)d(v,w)$. Note also that we have $d(v,w) \leq b_i d$ for any i and $(d,w) \in L_{i,v}(\mathcal{O}(n), n)$, as well as $i_{v,w} \leq i_{\max}$, because $\varepsilon d(v,w)/h_{v,w} \leq W_{\max}$. Consequently, for all $v, w \in V$ it holds that

$$d(v,w) \leq \tilde{d}(v,w) := \min_{i \in \{1,\ldots,i_{\max}\}} \{b_i d \mid (d,w) \in L_{i,v}(\mathcal{O}(n), n)\} \leq (1+\varepsilon)d(v,w).$$

As the G_i are unweighted graphs and rounding edge weights can be done locally, we can compute for each i the lists $L_{i,v}(\mathcal{O}(n), n)$ concurrently in $\mathcal{O}(n)$ rounds by Corollary 8.11; the virtual nodes "on" edges are simply simulated by one of the nodes incident to the corresponding edge in G. As ε is a constant,

$$i_{\max} = \lceil \log_{1+\varepsilon} W \rceil \in \mathcal{O}(\log W) \subseteq \mathcal{O}(\log n). \qquad \square$$

Remarks:

- One can use this rounding approach also to construct faster approximate solutions, but in order to reach the lower bound, we currently have to pay a factor of rougly $\log n$ in terms of approximation.

- You'll see some results on this in the exercises.

What to take home

- Sometimes a simplified version of the problem is worth studying, as the ideas turn out to be useful for more general cases, too.
- On the other hand, special cases may admit better solutions, and sometimes this is all we care about. For instance, in unweighted graphs one can solve APSP with small messages up to factor $\mathcal{O}(\log n)$ in $D \log^{\mathcal{O}(1)} n$ rounds. On weighted graphs, *any* algorithm that fast may hit the fan!
- If you want small messages: pipelining, pipelining, pipelining! Throw in some pipelining for good measure.

Chapter Notes

The almost linear lower bound for the APSP problem (without renaming) was shown independently and concurrently in two papers [Nan14, PSL13]. The exact unweighted APSP algorithm given here is from [LP13]. An elegant previous solution solving APSP in the same time was also given concurrently and independently in two papers [HW12, PRT12]. However, this algorithm requires $\Omega(n)$ time for computing the lists $L_v(h,k)$ for *any* $h > 0$, $k > 1$, and $|S|$. The paper by Holzer and Wattenhofer [HW12] contains a second algorithm that achieves running time $\mathcal{O}(h + |S|)$ for that task. This is ok for a large variety of applications, but if we have $S = V$, the algorithm is slow. For the fast APSP approximation with relabeling, we need the algorithm presented here.

The tree relabeling scheme in this lecture is nothing more than a composition of folklore results. In contrast, the *compact* (i.e., little-memory and small-labels) tree labeling scheme by Thorup and Zwick [TZ01] is more clever and at the heart of many compact routing schemes!

The rounding technique for transforming the ($(1+\varepsilon)$-approximate) weighted problem into a collection of unweighted problems was used by Nanongkai [Nan14] in the distributed context. However, it found earlier application for the centralized APSP problem [Zwi02], for which the fastest known algorithms are based on fast matrix multiplication.

Bibliography

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. 31st ACM Symp. on Principles of Distributed Computing*, 2012.

[LP13] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proc. 32nd ACM Symp. on Principles of Distributed Computing*, 2013.

[Nan14] Danupon Nanongkai. Distributed Approximation Algorithms for Weighted Shortest Paths. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 565–573, 2014.

[PRT12] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proc. 39th Int. Colloq. on Automata, Languages, and Programming*, 2012.

[PSL13] B. Patt-Shamir and C. Lenzen. Fast Routing Table Construction Using Small Messages [Extended Abstract]. In *Proc. 45th Symposium on the Theory of Computing (STOC)*, 2013.

[TZ01] Mikkel Thorup and Uri Zwick. Compact Routing Schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures*, 2001.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.