All Pairs Shortest Paths[1]

Seminar on Advanced Algorithms and Data Structures - Report

Fabian Gessler

Introduction and Overview

Before diving into the in the paper presented algorithms, we need to look at what the All Pairs Shortest Paths (APSP) problem is and how it relates to the All Pairs Almost Shortest Paths (APASP) problem. For this we first define some commonly used terms in graph theory.

terms in graph theory

(un)weighted graphs: A weighted graph is a graph which has a number (the weight) assigned to each edge. Here (if not stated otherwise) we only look at unweighted graphs, which means that the distance between two vertices connected by an edge is exactly one.

(un)directed graphs: A directed graph is a graph in which edges have an orientation, meaning we can only traverse the edge along its orientation. Here we only look at undirected graphs, which means that we can travel across an edge in both directions.

dominating set: A dominating set for a graph is a subset of vertices in the graph such that each vertex is either in the dominating set itself or adjacent to a vertex in the dominating set.

dominate(G, s): We run an algorithm on the graph G that constructs a dominating set D that dominates all the vertices with degree higher than s and also gives us an edge set E such that for each vertex with degree higher than s we have an edge that connects it to a vertex in D. Note that if we first compute D we can construct E easily by finding one neighbouring vertex for each high degree vertex that is in D and put the connecting edge into E. For all the high degree vertices that are in D themselves, we can just add any neighbouring vertex to D and the corresponding edge to E. This way we at worst double the size of D, which has no influence on the asymptotic running time or space.

Dijkstra's algorithm: Dijkstra's algorithm runs on a graph $G = (V, E)$ from a starting vertex s and finds distances and a tree of shortest paths from s to all other vertices of in G in $O(m + n \log n)$ for graphs with general weights.

δ : In a graph $\delta(a, b)$ denotes the shortest distance from vertex a to vertex b.

bfs$(G, \hat{\delta}, u)$: We run breadth-first search (bfs) on the graph G from starting vertex u. $\hat{\delta}$ is a matrix that contains estimated distances between the vertices in G and is updated during the run where shorter distances are found.

djikstra$(G, \hat{\delta}, u)$: We run Djikstra's algorithm on the graph G from starting vertex u. $\hat{\delta}$ is a matrix that contains estimated distances between the vertices in G and is updated during the run where shorter distances are found.

All Pairs Shortest Paths

The All Pairs Shortest Paths problem is - as the name suggests - the problem of computing the shortest paths for all pairs of vertices in a graph. For unweigthed graphs, we can compute this by

simply running breadth first search (bfs) on all vertices in overall $O(n(m+n))$ time. Even for dense graphs, m is bounded by $\binom{n}{2}$ which is in $O(n^2)$ and gives a bound for the runing time for APSP of $O(n^3)$.

All Pairs Almost Shortest Paths
As usual, if we don't find a way to improve the running time of a problem, we start to relax the requirements. So here, we only care about an approximation of the shortest paths. There are two metrics to say something about how good an instance of such an approximation is. The first one is the additive error, the second one the multiplicative error. Since we want almost shortest paths between to vertices u and v it makes sense to only allow distance approximations $\hat{\delta}(u,v)$ of the actual distance $\delta(u,v)$ with $\hat{\delta}(u,v) \geq \delta(u,v)$. To see how good such an approximation is we can use one of the two mentioned metrics to give us the guarantee that $\hat{\delta}(u,v) \leq \delta(u,v) + s$ or $\hat{\delta}(u,v) \leq \delta(u,v) \times f$ for a given summand s or factor f respectively. Distance approximations that have an error factor of f are called stretch f estimated distances, distance approximations that have an additive error of s are called surplus s estimated distances, which we will mostly be looking at.

Goal
This Papers main goal is to improve an algorithm from Aingworth et al. [2], which we won't discuss in detail here. It is sufficient to know that Aingowrth et al. describe an algorithm which solves the APASP problem with an additive error of 2 in time $\tilde{O}(n^{2.5}\sqrt{\log n})$ which is based on the following observation: There is a small set of vertices that dominates all the high degree vertices of a graph. Intuitively this makes sense: Vertices with a high degree have a lot of neighbours that can potentially be in the dominating set. So if we choose our threshold s for the degree of high degree vertices, then from the work done in [2] we can conclude the following Lemma:

Lemma 1 Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Let $1 \leq s \leq n$. A set D of size $O(\frac{(n \log n)}{s})$ that dominates all the vertices of degree at least s in the graph can be found deterministically in $O(m + ns)$ time.

APASP$_k$
We will simply present this algorithm family after having looked at the APASP$_2$ algorithm but not cover it deeply. The paper looks at a family of algorithms APASP$_k$ for $k \geq 2$ that solves the APASP problem. For each even $k > 2$ APASP$_k$ has running time $\tilde{O}(min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ and a one-sided error of at most k. At first, this seems unnecessarily complicated but when k increases, the running time decreases, so we see that this family exhibits a trade-off between running time and accuracy. With this, we can push the running time towards $\tilde{O}(n^2)$ by choosing $k = \Omega(\log n)$. This means we can compute APASP with an additive error of $\log(n)$ in $\tilde{O}(n^2)$ time. Also, we define APASP$_\infty$ as an instance of APASP$_k$ with $k = 2\lfloor \log n \rfloor$, which produces stretch 3 distances in $\tilde{O}(n^2)$ time.

We will be focusing on an APASP$_2$ algorithm that is the base case of the APASP$_k$ algorithms and improves the running time for the APASP problem with additive error 2 to $\tilde{O}(min\{n^{\frac{3}{2}}m^{\frac{1}{2}}, n^{\frac{7}{3}}\})$ from the previously achieved bound of $\tilde{O}(n^{2.5}\sqrt{\log n})$.

Algorithms

APASP$_2$
The APASP$_2$ algorithm consists of two algorithms apasp$_2$ and $\overline{\text{apasp}_3}$ with running time $\tilde{O}(n^{\frac{3}{2}}m^{\frac{1}{2}})$ and $\tilde{O}(n^{\frac{7}{3}})$ respectively. Both apasp$_2$ and $\overline{\text{apasp}_3}$ produce surplus 2 distances, so we just choose the faster one to get the running time of $\tilde{O}(min\{n^{\frac{3}{2}}m^{\frac{1}{2}}, n^{\frac{7}{3}}\})$ for APASP$_2$. We see that apasp$_2$ is faster for sufficiently sparse graphs ($m < n^{\frac{5}{3}}$) and $\overline{\text{apasp}_3}$ is faster for sufficiently dense graphs ($m > n^{\frac{5}{3}}$). The focus should lie on the functionality of the apasp$_2$ algorithm, as it has one less phase of vertex

partitioning and distance calculation. The $\overline{\text{apasp}_3}$ has the same idea in mind but we will look at it only after having thoroughly analyzed the apasp_2 algorithm.

apasp_2

Before we go into the apasp_2 algorithm, we want to provide some intuition of what is happening. We want to compute distances and we are okay with having an additive error of 2, but not more. The main idea is to first select a subset of vertices and simply compute their distances to all vertices in the graph. No error so far. For all other vertices, we construct a new edge set (that is not necessarily a subset of the edge set in the original graph) and compute their distances on this new edge set only. Why would this be a good idea? We see that this idea is only significantly faster than naively solving the APSP problem, if we make sure that in both phases we somehow have a smaller running time than the needed $O(n^3)$ for computing all distances. Also we have to make sure that the computed distances are actually good enough, meaning surplus 2 distances.

What is done in the algorithm is to first select the dominating set of all vertices with a degree higher than $(\frac{m}{n})^{\frac{1}{2}}$ to compute distances on, of which we know there exists a tighter bound than $O(n)$ (Lemma 1). This way we only need $O(f(n)(m+n)) = O(f(n)n^2)$ time to compute the distances, where $f(n)$ is the sub-linear bound obtained from the Lemma 1.
In the second step, we reduce the running time not by having a tighter bound than $O(n)$ on the vertices, but having a tighter bound than $O(n^2)$ for the edges. We need a special edge set that somehow has a tighter bound than $O(n^2)$, but still computes the distances (with an additive error of at most 2). Below is the apasp_2 algorithm from the paper that computes surplus 2 distances. We will have a look at what this edge set actually is and why we can say anything about its bound and error of estimated distances.
(Variables with index 1 are related to the high degree vertices, the ones with index 2 are related to the low degree vertices.)

Algorithm **apasp_2**:
input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\{\hat{\delta}(u,v)\}_{u,v}$ of estimated distances.
(1) $s_1 \leftarrow (\frac{m}{n})^{\frac{1}{2}}$
(2) $V_1 \leftarrow \{v \in V | \deg(v) \geq s_1\}$
(3) $E_2 \leftarrow \{(u,v) \in E | \deg(u) < s_1 \text{ or } \deg(v) < s_1\}$
(4) $(D_1, E^*) \leftarrow \textbf{dominate}(G, s_1)$

(5) For every $u, v \in V$ let $\hat{\delta}(u,v) \leftarrow \begin{cases} 1 & \text{if} (u,v) \in E, \\ +\infty & otherwise. \end{cases}$

(6) For every $u \in D_1$ run $\textbf{bfs}(G, \hat{\delta}, u)$
(7) For every $u \in V \backslash D_1$ run $\textbf{dijkstra}((V, E_2 \cup E^* \cup (\{u\} \times D_1)), \hat{\delta}, u)$

To get an idea of what is happening, we look at the following example:

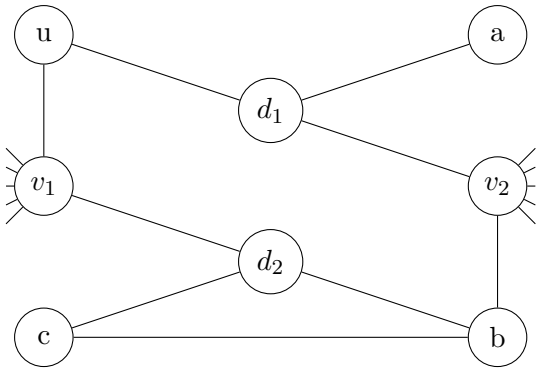

Figure 1a): Part of an example graph to visualise what apasp$_2$ does in the first distance computing phase (Line 6). This is just the original graph we got as input for apasp$_2$. We denote the vertices of high degree with v_1 and v_2 and vertices in the dominating set with d_1 and d_2. Indicated are some edges that connect this part to the rest of the graph.

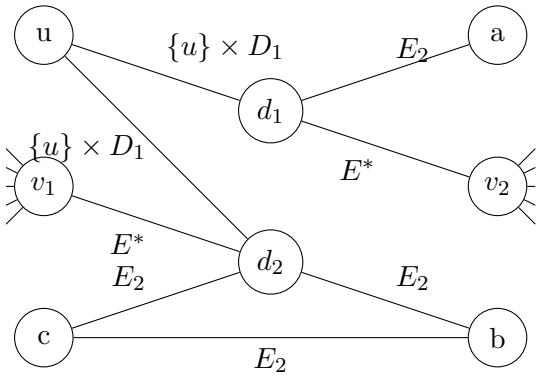

Figure 1b): Part of an example graph to visualise what apasp$_2$ does in the second distance computing phase (Line 7), starting from vertex u. We now consider not the edge set in the original graph but the one constructed in Line 7. We denote the vertices of high degree with v_1 and v_2 and vertices in the dominating set with d_1 and d_2. Next to each edge is indicated to which edge set this edge belongs to. Note that certain edges belong to multiple edge sets (e.g. (u,d1) is also in E_2). Indicated are some edges that connect this part to the rest of the graph, but we do not know which of these edges are actually in the considered edge set.

Now we get to the interesting part. How fast is this actually and why does it compute distances with an additive error of at most 2? We take a look at the analysis, first why it even works and then how fast it is.

accuracy:
The distances found in Line 6 are exact. For all distances we find in Line 7, there exist a corresponding path (not necessarily the shortest one) in the original graph with the same length, so $\delta(u,v) \leq \hat{\delta}(u,v)$. To further prove that $\delta(\hat{u,v}) \leq \delta(u,v)+2$ we have to get more technical and look at the following two (non-exclusive but exhaustive) cases:
Case 1: There is a shortest path between u and v that contains a vertex from V_1.
Let w be the last vertex on the path that belongs to V_1 (Figure 2). All the edges on the path from w to v touch vertices in V_2 and therefore belong to E_2. Let $w' \in D_1$ be such that $(w,w') \in E^*$. The edge (w,w') belongs to E^*. As $w' \in D_1$, the weighted edge (u,w') is in $\{u\} \times D_1$. The weight of this edge is $\delta(u,w')$, the distance between u and w' in G, found by the BFS from $w' \in D_1$. Note that $\delta(u,w') \leq \delta(u,w)+1$. By running Dijkstras algorithm from u, we find therefore that

$$\hat{\delta}(u,v) \leq \delta(u,w') + \delta(w',w) + \delta(w,v) \leq (\delta(u,w)+1) + 1 + \delta(w,v) = \delta(u,v) + 2.$$

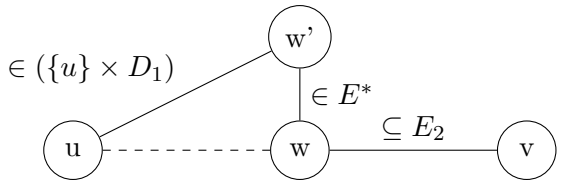

Figure 2: Case 1 of the apasp$_2$ accuracy analysis. The line connecting w and v represents an arbitrary number of edges.

Case 2: There is a shortest path between u and v that does not contain any vertex from V_1. This shortest path is contained in (V, E_2) and therefore $\hat{\delta}(u, v) = \delta(u, v)$.

Note that these two cases are not exclusive only because we can have multiple shortest paths in a graph.

running time:
Line 1-3: We simply run through all vertices and edges which only needs $O(n + m)$ time (trivial).
Line 4: We can simply use s_1 as the threshold of vertex degrees in Lemma 1, which gives us a dominating set of size $O((n^{\frac{3}{2}}/m^{\frac{1}{2}}) \log n) = \tilde{O}((n^{\frac{3}{2}}/m^{\frac{1}{2}})$. Here is where the $\tilde{O}$ comes in and the logarithmic term from the Lemma vanishes. Lemma 1 states further, that we then can find this dominating set in $O(m + n(s_1)) = O(m + (mn)^{\frac{1}{2}}) = O(n^2)$ time.
Line 5: We set entries of a $n \times n$ matrix which is done in $O(n^2)$ (trivial).
Line 6: A single bfs on any graph runs in $O(m+n) = O(m)$. Since we here run bfs on $|D_1| = O(n^{\frac{3}{2}}/m^{\frac{1}{2}})$ vertices we need $O((n^{\frac{3}{2}}/m^{\frac{1}{2}})m) = O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ time.
Line 7:

Lemma 2 (without proof): If the weights of the edges are integers in the range $\{1, 2, \cdots, n\}$ and no distance is higher than m, the djikstra algorithm can be implemented to run in $O(m + n)$ time.

Here we run dijkstra in a weighted graph. All edges have weight one, except the edges in $(\{u\} \times D_1)$ which have the weight of the estimated distance in $\hat{\delta}$. This is a graph which has edge weights in the range $\{1, 2, \cdots, n\}$ and no distance higher than m (because the original graph is unweighted and has m edges with weight 1), so a single run of dijkstra can be done in $O(m + n)$ time (Lemma 2). The amount of low degree vertices has no tighter bound than $O(n)$ so we have to run dijkstra $O(n)$ times. We do not run this on the original graph G, but a graph with the same vertices and this new edge set that should have a tighter bound than $O(n^2)$. So far we managed to run the algorithm in $O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ from Line 6, so if we are able to show that our edge set is in size $O((nm)^{\frac{1}{2}})$, we get an overall running time of $O((nm)^{\frac{1}{2}}n) = O(n^{\frac{3}{2}}m^{\frac{1}{2}})$. For this we need to show that each individual edge set of the union used for dijkstra is bounded by $O((nm)^{\frac{1}{2}})$ for the union to also be bounded by this.
$\mathbf{E}_2$: Each low degree vertex has less than $(\frac{m}{n})^{\frac{1}{2}}$ outgoing edges, meaning the number of edges touching a low degree vertex is bounded by $O(n(\frac{m}{n})^{\frac{1}{2}}) = O((nm)^{\frac{1}{2}})$.
$\mathbf{E}^*$: The call to **dominate** gives us this edge set in size $O(n)$.
$\mathbf{(\{u\} \times D_1)}$: In each run, u contains exactly one vertex, so this is in $|D_1| = O(n^{\frac{3}{2}}/m^{\frac{1}{2}})$
In conclusion, all edge sets are small enough for the algorithm to run in $O(n^{\frac{3}{2}}m^{\frac{1}{2}})$.

$\overline{\text{apasp}_3}$
Below is the $\overline{\text{apasp}_3}$ algorithm from the paper that also computes surplus 2 distances. We will not look into this algorithm deeply, the details are still written here. We mention only the main differences to the apasp$_2$ algorithm: The approach is similar but we now have 3 phases of computing distances (Lines 7-9), in each considering a different vertex and edge set, which leads to a different running time

bound of $O(n^{\frac{7}{3}})$.
Again, to proof that this works we would have to make sure that in each phase we have a significantly smaller vertex or edge set to be faster than trivially computing the distances exactly.

Algorithm $\overline{\textbf{apasp}_3}$:
input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\hat{\delta}(u, v)_{u,v}$ of estimated distances.
(1) $s_1 \leftarrow n^{\frac{2}{3}}$; $s_2 \leftarrow n^{\frac{1}{3}}$
(2) For $i \leftarrow 1$ to 2 let $V_i \leftarrow \{v \in V | \deg(v) \geq s_i\}$
(3) For $i \leftarrow 2$ to 3 let $E_i \leftarrow \{(u, v) \in E | \deg(u) < s_{i-1}$ or deg $(v) < s_{i-1}\}$
(4) For $i \leftarrow 1$ to 2 let $(D_i, E_i^*) \leftarrow \textbf{dominate}(G, s_i)$
(5) $E^* \leftarrow E_1^* \cup E_2^*$

(6) For every $u, v \in V$ let $\hat{\delta}(u, v) \leftarrow \begin{cases} 1 & \text{if} (u, v) \in E, \\ +\infty & otherwise. \end{cases}$

(7) For every $u \in D_1$ run $\textbf{bfs}(G, \hat{\delta}, u)$
(8) For every $u \in D_2$ run $\textbf{bfs}((V, E_2), \hat{\delta}, u)$
(9) For every $u \in V$ run $\textbf{dijkstra}((V, E_3 \cup E^* \cup (D_1 \times V) \cup (D_2 \times D_2) \cup (\{u\} \times D_2)), \hat{\delta}, u)$

Again the question is why would this algorithm run in the promised running time $O(n^{\frac{7}{3}})$ and still compute surplus 2 distances. This time we can profit from the work already done in apasp_2 but there are some changes in the algorithm we need to look at. Again we start with the accuracy then move to the time complexity.

accuracy:
As in apasp_2, it is clear that $\delta(u, v) \leq \hat{\delta}(u, v)$ and we only need to show that $\delta(\hat{u, v}) \leq \delta(u, v) + 2$, so we look at the following three cases in detail:
Case 1: There is a shortest path between u and v that contains a vertex w from V_1.
Let $w' \in D_1$ such that $(w, w') \in E^*$ (Figure 3). The edges (u, w') and (w', v) belong to the graph on which Dijkstras algorithm is run from u. The weights of these edges are $\delta(u, w')$ and $\delta(w', v)$, the distances found by the BFS on G from $w' \in D_1$. Note that $\delta(u, w) \leq \delta(u, w) + 1$ and $\delta(w', v) \leq 1 + \delta(w, v)$. By running Dijkstras algorithm from u, we find therefore that

$\hat{\delta}(u, v) \leq \delta(u, w,) + \delta(w', v) \leq \delta(u, v) + 2.$

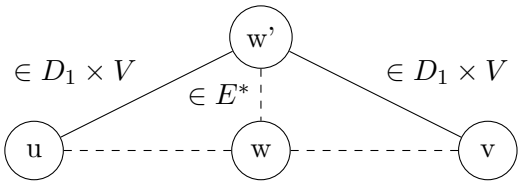

Figure 3: Case 1 of the accuracy analysis of $\overline{\text{apasp}_3}$

Case 2: There is a shortest path between u and v that contains vertices from V_2 but not from V_1.
This case is very similar to case 1 in the accuracy proof of apasp_2. Let w be the last vertex on the path that belongs to V_2. All the edges on the path from w to v touch vertices in V_3 and are therefore in E_3. Let $w' \in D_2$ be such that $(w, w') \in E^*$. The graph we run djikstra on contains weighted edges connecting u to all the vertices of D_2. It contains in particular a weighted edge (u, w'). The weight of this edge is the distance $\delta_2(u, w')$ between u and w' in the graph $G_2 = (V, E_2)$, found by the BFS

from $w' \in D_2$ on G_2. As all the edges on the path from u to w, as well as (w, w') belong to E_2, we get that $\delta_2(u, w') \leq \delta(u, w) + l$. By running Dijkstras algorithm from u we find therefore that

$\hat{\delta}(u, v) \leq \delta_2(u, w') + \delta(w', w) + \delta(w, v) \leq \delta(u, v) + 2.$

Case 3: There is a shortest path between u and v that does not contain any vertex from V_2.
This shortest path is then contained in (V, E_3) and therefore $\hat{\delta}(u, v) = \delta u, v$.

running time:
Line 1-3: Here we split the vertices into 3 instead of 2 classes according to their degree and accordingly have multiple edge sets touching the medium/low degree vertices. For this we still simply go over all vertices and edges in $O(n + m)$ (trivial). Note that $V_1 \subseteq V_2$ and $E_2 \subseteq E_3$.
Line 4-5: Here we dominate the different vertex sets separately and get two different dominating sets. According to Lemma 1, these two calls to **dominate** produce dominating sets of size $O(n \log n/n^{\frac{2}{3}}) = O(n^{\frac{1}{3}} \log n)$ and $O(n \log n/n^{\frac{1}{3}}) = O(n^{\frac{2}{3}} \log n)$ in time $O(m+n(n^{\frac{2}{3}})) = O(m+n^{\frac{5}{3}})$ and $O(m+n(n^{\frac{1}{3}})) = O(m + n^{\frac{4}{3}})$. Because O(m) is bounded by $O(n^2)$ we need only $O(n^2)$ time to compute D_1 and D_2. Note that E^* still has the two important properties of E_1^* and E_2^*; it is in size $O(n)$ and for each $u \in (V_1 \cup V_2) = V_2$ there exists $v \in D_1 \cup D_2$ such that $(u, v) \in E^*$.
Line 6: same as in Line 5 in apasp$_2$.
Line 7: Same as in Line 6 in apasp$_2$ but on a different set of starting vertices. We can run bfs in $\tilde{O}(n^{\frac{1}{3}} \log n(m + n)) = \tilde{O}(n^{\frac{1}{3}} n^2) = \tilde{O}(n^{\frac{7}{3}})$ time.
Line 8: Here we not only have a different starting vertex set but also we restrict the edges in the graph to be used. The restricting of the edges lets us have a tighter bound on the number of edges involved, namely $O(n^{\frac{5}{3}})$ since E_2 consists of all the edges that touch a vertex of degree lower than s_1 of which there can be at most $ns_1 = n^{\frac{5}{3}}$. Considering this, these calls to **bfs** also only take $O(n^{\frac{2}{3}} n^{\frac{5}{3}}) = O(n^{\frac{7}{3}})$ time.
Line 9: same as in Line 7 in apasp$_2$ but we have a different edge set. If the edge set is in $O(n^{\frac{4}{3}})$ we get our running time of $\tilde{O}(n^{\frac{7}{3}})$.
$\mathbf{E_3}$: These are the edges that touch vertices of degree smaller than s_2 so there can only be at most $s_2 * n = O(n^{\frac{4}{3}})$ of them.
$\mathbf{E^*}$: $E^* = E_1^* \cup E_2^*$ and E_1^* and E_2^* are in $O(n)$, so E^* is also in $O(n)$.
$\mathbf{D_1 \times\ V}$: $|D_1| = O(n^{\frac{1}{3}} \log n)$ and $|V| = O(n)$, so $|(D_1 \times V)| = O(n^{\frac{4}{3}})$.
$\mathbf{D_2 \times\ D_2}$: $|D_2| = O(n^{\frac{2}{3}})$, so $|D_2 \times D_2| = O(n^{\frac{4}{3}})$.
$\mathbf{\{u\} \times\ D_2}$: $|\{u\}| = 1$ and $|D_2| = O(n^{\frac{2}{3}})$, so $\{u\} \times D_2 = O(n^{\frac{2}{3}})$.
In conclusion, each edge set is in $O(n^{\frac{4}{3}})$ and $\overline{\text{apasp}_3}$ actually runs in $O(n^{\frac{7}{3}})$

Summary so far
We showed an APASP$_2$ algorithm that consists of two algorithms that each compute surplus 2 distances faster than in $O(n^3)$. Based on the density of the graph we work on we can pick either apasp$_2$ or $\overline{\text{apasp}_3}$ to get the minimum of both running times as our overall running time.

APASP$_k$
Note: We will not present the following parts as it would take to much time. Since they still are a big part of the paper we decided to include them here anyway.

After seeing the APASP$_2$ algorithm consisting of the two parts apasp$_2$ and $\overline{\text{apasp}_3}$ with additive error of 2, APASP$_k$ reduces the running time by allowing larger errors up to k.
It again consists of two algorithms apasp$_k$ and $\overline{\text{apasp}_k}$, but we have have for a fix k an error bound of $2(k-1)$ in apasp$_k$ and an error bound of $2(\lfloor \frac{k}{3} \rfloor + 1)$ in $\overline{\text{apasp}_k}$. If we want to get APASP$_k$ that has an additive error of k, we need to look at apasp$_{\frac{k}{2}+1}$ and $\overline{\text{apasp}}_{\frac{(3k-2)}{2}}$ and choose the faster one to be

in the running time $\tilde{O}(min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$. The pseudo code for the algorithms is below, the proofs to the running time and accuracy is very similar to the ones found for apasp$_2$ and $\overline{\text{apasp}_3}$, as they are just special cases of these more general algorithms, so they are omitted.

Algorithm **apasp**$_k$:
input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\hat{\delta}(u, v)_{u,v}$ of estimated distances.
(1) For $i \leftarrow 1$ to $k-1$ let $s_i \leftarrow (\frac{m}{n})^{1-\frac{i}{k}}$
(2) For $i \leftarrow 1$ to $k-1$ let $V_i \leftarrow \{v \in V | \deg(v) \geq s_i\}$
(3) For $i \leftarrow 2$ to k let $E_i \leftarrow \{(u, v) \in E | \deg(u) < s_{i-1}$ or deg $(v) < s_{i-1}\}$
(4) For $i \leftarrow 1$ to $k-1$ let $(D_i, E_i^*) \leftarrow$ **dominate**(G, s_i)
(5) $E_1 \leftarrow E$; $D_k \leftarrow V$; $E^* \leftarrow \cup_{1 \leq i < k} E_i^*$

(6) For every $u, v \in V$ let $\hat{\delta}(u, v) \leftarrow \begin{cases} 1 & \text{if}(u, v) \in E, \\ +\infty & otherwise. \end{cases}$

(7) For every $i \leftarrow 1$ to k do
(8) For every $u \in Di$ run dijkstra$((V, E_i \cup E^* \cup (\{u\} \times V)), \hat{\delta}, u)$

Algorithm $\overline{\textbf{apasp}_k}$:
input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\hat{\delta}(u, v)_{u,v}$ of estimated distances.
(1) For $i \leftarrow 1$ to $k-1$ let $s_i \leftarrow n^{1-\frac{i}{k}}$
(2) For $i \leftarrow 1$ to $k-1$ let $V_i \leftarrow \{v \in V | \deg(v) \geq s_i\}$
(3) For $i \leftarrow 2$ to k let $E_i \leftarrow \{(u, v) \in E | \deg(u) < s_{i-1}$ or deg $(v) < s_{i-1}\}$
(4) For $i \leftarrow 1$ to $k-1$ let $(D_i, E_i^*) \leftarrow$ **dominate**(G, s_i)
(5) $E_1 \leftarrow E$; $D_k \leftarrow V$; $E^* \leftarrow \cup_{1 \leq i < k} E_i^*$

(6) For every $u, v \in V$ let $\hat{\delta}(u, v) \leftarrow \begin{cases} 1 & \text{if}(u, v) \in E, \\ +\infty & otherwise. \end{cases}$

(7) For every $i \leftarrow 1$ to k do
(8) For every $u \in Di$ run dijkstra$((V, E_i \cup E^* \cup (\{u\} \times V) \cup (\cup_{i+j1+j2 \leq 2k+1} D_{j1} \times D_j 2)), \hat{\delta}, u)$

Boolean Matrix Multiplication

Theorem 1: If all the distances in an undirected n vertex graph can be approximated with a one-sided error of at most one in $O(A(n))$ time, then Boolean matrix multiplication can also be performed in $O(A(n))$ time.

Proof: Let $C = A \times B$ be a Boolean matrix multiplication of size $n \times n$. We can also interpret $A \times B$ as the graph $G = (V, E)$ with

$V = \{u_1, \cdots, u_n\} \cup \{v_1, \cdots, v_n\} \cup \{w_1, \cdots, w_n\}$,
$E = \{u_1, v_k | a_{ik} = 1\} \cup \{v_k, w_j | b_{kj} = 1\}$.

In this case, a_{ik} indicates whether there is an edge between the vertex u_i and v_k and b_{kj} indicates whether there is an edge between the vertex v_k and w_j. In the result, $c_i j = 1$ if and only if there exists a direct path (u_i, v_k, w_j) for some v_k and therefore $\delta_G(u_i, w_j) = 2$. So if all the distances in an undirected n vertex graph can be calculated in $O(A(n))$ time, then Boolean matrix multiplication can also be performed in $O(A(n))$ time. But we don't actually need to compute the distances exactly, as the only

condition we need to check in G is, whether $\delta_G(u_i, w_j) \leq 3$, since $\delta_G(u_i, w_j) \in \{2, 4, 6, \cdots\} \cup \{\infty\}$ and the Theorem holds.

Summary

We get algorithms with flexible accuracy and running time, that, in the case of an additive error of 2, is faster then the previously explored algorithm in [2]. Also important to mention is that the APASP_∞ algorithm runs in quadratic time and produces constant stretch estimated distances (with a factor of 3), which was not previously achieved. The work on Boolean Matrix Multiplication shows that if we want to further improve the error estimate to a surplus 1 distance we have to look into the lower bounds of boolean matrix multiplication.

References

[1] Dorit Dor, Shay Halperin, Uri Zwick. *All Pairs Almost Shortest Paths.* FOCS 1999.

[2] D. Aingworth, C.Chekuri, and R. Motwani. *Fast esimation of diameter and shortest paths (without matrix multiplication).* In Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, Georgia, pages 547-553, 1996