An Optimized All pair Shortest Paths Algorithm

Vijay Shankar Pandey
Board of Apprenticeship Training (SR), Chennai (TN)

Rajendra Kumar
Vidya College of Engineering, Baghpat Road, Meerut (UP)

Dr. P K Singh
MMM Engineering College, Gorakhpur (UP)

ABSTRACT

In this paper, we present an algorithm to compute all pairs optimized shortest paths in an unweighted and undirected graph with some additive error of at most 2.This algorithm can be extended for weighted graph also but it will not work for directed graph due to absence of commutative property. The algorithm runs in $O(n^{5/2})$ times, where n is the number of vertices in the graph. This algorithm is much simpler than the existing algorithms. A study of upper bounds on the size of a maximal independent set of such graphs has been performed.

Keywords
Optimized, Commutative, Sub-Cubic, Additive Error, Multiplicative Error

1. INTRODUCTION

Single source shortest paths algorithm gives the shortest paths from a source vertex to every other vertex of the graphs. One such classical algorithm for unweighted graphs is breadth-first search (BFS) algorithm [8]. Some algorithms do not use single source shortest paths algorithms as subroutine, like one of the most classical algorithm for directed and weighted graph by Floyd and Warshall [8]. An all pairs optimized shortest paths algorithm does not report the exact shortest paths for every pair of vertices and distance reported may have some error.

Almost every algorithm for the all pairs shortest paths problem, except those based on fast matrix multiplication, has running time of $O(n^3)$ in worst case. There exist algorithms based on fast matrix multiplication for the all pairs shortest paths problem that achieve sub-cubic running time, but these fast matrix multiplication algorithms are better than the naive $O(n^3)$ time algorithm only for very large values of n. This is where the need of approximation algorithms arises. Though optimization algorithms do not give precise output, but are faster. Many sub-cubic running time and simple algorithms have been designed for all pairs optimized shortest path problem. These algorithms achieve sub-cubic running time, but the distance reported has some multiplicative or additive error.

Definition 1: An algorithm is said to compute all pairs α-optimized shortest paths or all pairs optimized shortest paths of stretch α for some $\alpha \geq 1$, if for every pair of vertices $u, v \in V$, the distance reported is bounded by $\alpha.\delta(u, v)$, where $\delta(u, v)$ is the actual shortest distance between u and v.

Definition 2: An algorithm is said to compute all pairs shortest paths with an additive one-sided error of at most β or all pairs optimized shortest paths of surplus β for some $\beta \geq 0$, if for every pair of vertices $u, v \in V$, the distance reported is bounded by $\delta(u, v) + \beta$, where $\delta(u, v)$ is the actual shortest distance between u and v.

An interesting property of the shortest path is that a shortest path between two vertices consists of other shortest paths. This optimal substructure property of shortest path is used by every shortest path algorithm. For an unweighted graph, following lemma states the optimal substructure property of shortest paths more precisely.

Lemma 1. Given an unweighted graph $G = (V, E)$, let path $p = (v_1, v_2, \ldots, v_k)$ be a shortest path from a vertex v_1 to a vertex v_k and, let $p_{ij} = (v_i, v_{i+1}. \ldots . v_j)$ be the subpath of p, for any i and j such that $1 \leq i \leq j \leq k$. Then p_{ij} is a shortest path from v_i to v_j.

Proof: Let length(p) be the length of path p. If we decompose path p into path from v_1 to v_i, path from v_i to v_j and path from v_j to v_k, then we have

$$\text{length}(p) = \text{length}(p_{1i}) + \text{length}(p_{ij}) + \text{length}(p_{jk}).$$

If there is a shorter path p'_{ij} than p_{ij} from v_i to v_j, then we have a path from v_1 to v_k, which is path p_{1i} followed by p'_{ij} followed by p_{jk}. The length of this path

$$\text{length}(p_{1i}) + \text{length}(p'_{ij}) + \text{length}(p_{jk}) < \text{length}(p),$$

since $\text{length}(p'_{ij}) < \text{length}(p_{ij})$. This contradicts the assumption that P is a shortest path from v_1 to v_k.

2. PREVIOUS WORK: ALL PAIRS SHORTEST PATH ALGORITHMS

One of the most classical algorithm for computing all pairs shortest paths is F1oyd-Warshall algorithm [8], which runs in $O(n^3)$ time. Floyd-Warshall algorithm uses the technique of dynamic programming. Johnson's algorithm [16] take $O(mn + n^2 \log n)$ time to compute the shortest paths between all pairs, and hence it is asymptotically better for space graphs, but it requires no negative cycle to be present in the graph. It uses a single source shortest paths algorithm as subroutine and runs it for all the vertices. However, in worst case when $m = O(n^2)$ it still takes $O(n^3)$ time. The all pairs shortest paths problem for directed graphs with nonnegative edge weights is closely related to the distance product of two matrices.

If $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$, matrices, the distance product $A \times B$ is the $n \times n$, matrix whose $(i, j)^{th}$ element is $(AB)_{ij} = \min_k\{a_{ik} + b_{kj}\}$. Fredman gave an $O(n^3(\log \log n/\log n)^{1/3})$ running time algorithm [11] to compute the distance products of two $n \times n$ matrices, whose bound was later improved to $O(n^3(\log \log n / \log n)^{1/2})$ by Takaoka [22]. For the same type of graphs (nonnegative edge weights), Karger et at. [17] and [19] gave an $O(m^* n + n^2 \log n)$ running time algorithm, where m^* denotes the number of edges in the essential subgraph H of the input graph $G = (V, E)$. The essential subgraph contains an edge $(u, v) \in G$ if that edge is uniquely the least-cost path between its vertices. However, in the worst case m^* can be as large as m.

For graphs with integer edge weights, Hagerup gave an

$O(mn+n^2 \log\log n)$ running time algorithm [14]. For undirected graphs, Pettie and Ramachandran gave an $O(mn\ \alpha(m, n))$ running time algorithm [20], where $\alpha(m, n)$ is Tarjan's inverse-Ackermann function. For undirected graphs with integer edge weights, Thorup [23 24] gave an $O(mn)$ running time algorithm. It uses a single source shortest paths algorithm, which bypasses the sorting bottleneck of Dijkstra's algorithm [9] and runs in $O(m)$ time.

Directed	Weight	Complexity	Ref.
yes	real	n^3	[8]
yes	real	$mn+n^2 \log n$	[16]
yes	real+	$n^3(\log\log n/\log n)^{1/2}$	[11], [22]
yes	real+	$m^* n+n^2 \log n$	[17], [19]
yes	integer	$mn+n^2 \log\log n$	[14]
no	real	$mn\alpha(m, n)$	[20]
no	integer	mn	[23, 24]

Table 1: All pairs shortest paths algorithm

Recent work by Alon, Galil, and Margalit [2], Alon and Naor [3], Galil and Margalit [12, 13], and Seidel [21] have shown that if matrix multiplication can be performed in $O(M(n))$ time, then the all pairs shortest paths problem for unweighted directed graphs can be solved in $\tilde{O}(\sqrt{n^3}\, M(n))$ time and the all pairs shortest paths problem for unweighted undirected graphs can be solved in $\tilde{O}(M(n))$ time. Here $\tilde{O}(f)$ is to hide the polylogarithmic factor i.e. $\tilde{O}(f)$ means $O(f \text{ polylog } n)$. The current best upper bound on matrix multiplication is $M(n) = O(n^{2.376})$ by Coopersmith and Winograd [7].

3. ALL PAIRS OPTIMIZED SHORTEST PATHS ALGORITHMS

[1] gave algorithm for all pairs shortest paths with additive error for unweighted and undirected graphs. They showed that surplus 2 estimate of the distance between k specified pairs of vertices can be computed in $O(n^{3/2}(k\log n)^{1/2})$. Time, i.e. surplus 2 estimates of all the shortest paths in the graph can be computed in $O(n^{5/2}(\log n)^{1/2})$ time. They also gave a 2/3-approximation algorithm for the diameter of a weighted and directed graph that runs in $O(m(n \log n)^{1/2} + n^2 \log n)$ time. Dor et al. [10] improved the results of Aingworth et al., and gave $\tilde{O}(n^{3/2}m^{1/2})\tilde{O}(n^{7/3})$ running time algorithms for all pairs shortest paths with additive error of at most 2. They also showed that all pairs shortest paths with additive error of at most k can be computed in $\tilde{O}(n^{2-1/k} m^{1/k}, n^{2+1/(3k-4)})$ time. For the weighted graphs Cohen and Zwick [6] adapted the technique of Dor et al. and obtained stretch 2 estimates of al shortest paths in $\tilde{O}(n^{3/2} m^{1/2})$ time, stretch 7/3 estimates in $\tilde{O}(n^{7/3})$ time and stretch 3 estimates in $\tilde{O}(n^2)$ time.

Weight	Error	Time*	Ref.
No	Surplus 2	$n^{5/2}$	[1]
No	Surplus 2	$n^{3/2} m^{1/2}$, $n^{7/3}$	[10]
No	Surplus 4	$n^{5/3} m^{1/3}$, $n^{11/5}$	[10]
No	Surplus $2(k-1)$	$n^{2-1/k} m^{1/k}$, $n^{2+1/(3k-4)}$	[10]
yes	Stretch 2	$n^{3/2} m^{1/2}$	[6]
yes	Stretch 7/3	$n^{7/3}$	[6]
yes	Stretch 3	n^2	[6]

*Ignoring polylogarithmic factors

Table 2: All pairs optimized shortest paths algorithm

For unweighted and undirected graphs Baswana et al. [4] gave randomized algorithms for computing all pairs nearly 2-optimized shortest paths. One algorithm takes expected $O(m^{2/3} n \log n + n^2)$ time and reports the distance not more than $2\delta(u, v)+1$ between any two vertices u and v. Another algorithm takes expected $O(n^2 \log^{3/2} n)$ time and reports the distance not more than $2\delta(u, v) + 3$. They also obtained an expected $O(n^2)$ time algorithm to compute all pairs 3-optimized distances in a graph. The work of Aingworth et al. [1] and Dor et al. [10] for all pairs shortest paths with additive error of at most 2, is based upon the simple observation that there is a small set of vertices that dominates all the high degree vertices of graph. A set of vertices D dominates a set u of vertices when every vertex in u has a neighbor in D. Dor et al. have shown that there exists a set D of size $O((n \log n)/s)$ such that $1 \le s \le n$, that dominates all the vertices of degree at least s in the graph and can be found deterministically in $O(m+n)$ time. To achieve the sub-cubic running time, Dor et al. run BFS on the input graph for $\tilde{O}(n^{3/2} /m^{1/2})$ vertices only, and for the rest of the vertices they run Dijkstra's algorithm [9] on a weighted graph with $O((nm)^{1/2})$ edges. Let $G = (V, E)$ be an undirected and unweighted graph. They partition the set of vertices into low degree vertices (degree less than $(m/n)^{1/2}$) and high degree vertices (degree at least $(m/n)^{1/2}$). Then they find a set D of size $\tilde{O}(n^{3/2}/m^{1/2})$ that dominates high degree vertices, in $O(m+n)$ time. They also construct an edge set $E' \subseteq E$ of size $O(n)$ such that for every high degree vertex u there exist a vertex $v \in D[t]$ such that $(u, v) \in E'$. First the BFS is performed on G for every vertex $u \in D$, which gives the shortest distance of u from every other vertex. Then for every vector $u \in V/D$ a weighted graph $G'(u)$ is constructed, which includes all the edges that touch low degree vertices, edges in E', and the edges connecting the vertex u with all the vertices of dominating set. The weight of an edge (u, v) for every $v \in D$ is equal to the shortest distance between u and v in G, found by the BFS that started at v, while the weight of the rest of the edges is 1. The number of edges in graph $G'(u)$ is $O(nm)^{1/2}$). Finally Dijkstra's algorithm is run, from every vertex $u \in V/D$ on the graph $G'(u)$. For every pair of vertices $u, v \in V/D$, the distance computed between u and v has an additive error of at most 2.

4. OUR APPROACH

To gain sub-cubic running time, instead of running BFS for each vertex of the graph, we can BFS for few vertex of the graph, and use the information obtained to compute the optimized all pairs shortest paths for the entire graph. Let $G = (V, E)$ be an unweighted and undirected input graph. We divide the vertices into three mutually exclusive and exhaustive sets: P, Q, and R. Then the BFS is run for each vertex in P. The knowledge of single source shortest paths for every vertex in P is then used to compute the optimized shortest paths for the vertices in Q and R.

Constructing *P*, *Q*, and *R*

To divide set of vertices V into P, Q, and R we use a simple greedy approach. We choose one of the maximum degree vertex and put this vertex into the set P and all of its neighbors into the set Q. Then from the graph we delete this vertex and all of its neighbors including the edges incident on these vertices. We repeat this procedure until we get a sub graph G' of G, with less than or equal to $n\sqrt{n}$ edges. All the vertices of G' constitute R.

Lemma 2. Set P contains less than $\sqrt{n}$ vertices.

Proof: We proof this by contradiction. We stop constructing P as soon as the number of edges in the sub graph G' (Obtained after deleting from G, the vertices in P

and Q and the edges incident on these vertices) becomes less than $n\sqrt{n}$. Suppose even after selecting $\sqrt{n}$ special vertices, the number of edges in the subgraph G' is more than $n\sqrt{n}$. It implies that at each iteration the number of edges in subgraph were more than $n\sqrt{n}$ because we do not add any edge anytime. Suppose $G_i = (V_i, E_i)$ is the subgraph at i^{th} iteration. Thus we have, for $1 \le i \le \sqrt{n}$,

$|E_i| \ge n\sqrt{n}$

If $d(v)$ is the degree of vertex v, then for $1 \le i \le \sqrt{n}$,

$$\sum d(v) \ge 2n\sqrt{n}, \quad v \in V_i$$

If $\alpha(V)$ is the average degree of the set of vertices V , then for $1 \le i \le \sqrt{n}$,

$$\alpha(V_i) \ge \sum d(v)/n \ge 2\sqrt{n} \text{ and } v \in V_i$$

Since at each iteration, we choose the vertex with maximum degree as the special vertex, it must have the degree more than or equal to the average degree $2\sqrt{n}$ .Since we delete the chosen special vertex and its neighbors at each iteration, it implies that we deleted more than $2\sqrt{n}$ vertices at each iteration. Thus after $\sqrt{n}$ iterations more than $2n$ vertices were deleted. This is a contradiction, as the graph consists of n vertices.

Computing shortest distance

After constructing the sets P, Q and R, we compute the shortest distances for vertices in the three sets as follows:

1. For each vertex $u \in P$ we compute and store the shortest distance from every vertex $v \in V$, using the BFS with u as root. This would take $m\sqrt{n}$ running time, which is sub-cubic $O(n^{5/2})$ in worst case.
2. For the vertices in Q if $s \in P$ is a neighbor of $u \in Q$, then we store the shortest distance between u and v as $\delta'(u, v) = \delta'(s, v) + 1$.The shortest distance $\delta(s, v)$ between s and v, is known accurately by the BFS that started at s. If $v \in Q$, then we store the shortest distance between u and v as $\delta'(u, v) = \min\{\delta(s, v), \delta'(s', u)\} + 1$; where $s' \in P$ is a neighbor of v. It can be shown that $\delta(u, v) \le \delta'(u, v) \le \delta(u, v) + 2$.
3. The sub-graph $G' = (V', E')$ of $G = (V, E)$ induced by the vertices in R has at most $n\sqrt{n}$ edges. For every vertex u, $v \in R$ we compute the shortest distance $\delta''(u, v)$ in the subgraph G' using the BFS for each vertex. This would take at most $|R|*|E'| = n \times n\sqrt{n}$ running time, which is $O(n^{5/2})$. If G' is not a connected graph, there may not exist any path between u and v. In this case, we store the shortest distance $\delta''(u, v)$) between u and v in G', as ∞. Now we store the shortest distance between u and v as $\delta'(u, v) = \min\{\delta''(u, v), \min_{s \in S}\{\delta(s, u) + \delta(s, v)\}\}$. It can be shown that $\delta(u, v) \le \delta'(u, v) \le \delta(u, v) + 2$.

Lemma 3. If $u \in Q$ and $v \in V/P$, there exists $\delta(u, v) \le \delta'(u, v) \le \delta(u, v) + 2$.

Proof: As $u \in Q$, let $s \in P$ be its neighbor. We have only one of the following relation between $\delta(u, v)$and $\delta(s, v)$:

(i) $\delta(u, v) > \delta(s, v)$
(ii) $\delta(u, v) < \delta(s, v)$
(iii) $\delta(u, v) = \delta(s, v)$

Case 1. $\delta(u, v) > \delta(s, v)$ consider the path between u and v, which consist of the shortest path between s and v of length $\delta(s, v)$ and the edge (u, s). The length of this path is $\delta(s, v) + 1$. Since no path between u and v can be shorter than the shortest path between u and v, $\delta(s, v) + 1 \ge \delta(u, v)$. But as $\delta(u, v) > \delta(s, v)$, we have $\delta(s, v) + 1 \ge \delta(u, v) > \delta(s, v)$. Thus $\delta(u, v) = \delta(s, v) + 1$ and in a in shortest path from u to v, first hop is s as shown in figure 1.

Case 2. This case is same as previous case, except that the roles of u and s are interchanged. Hence if $\delta(u, v) < \delta(s, v)$, $\delta(s, v) = \delta(u, v) + 1$ and in a shortest path from s to v first hop is u.

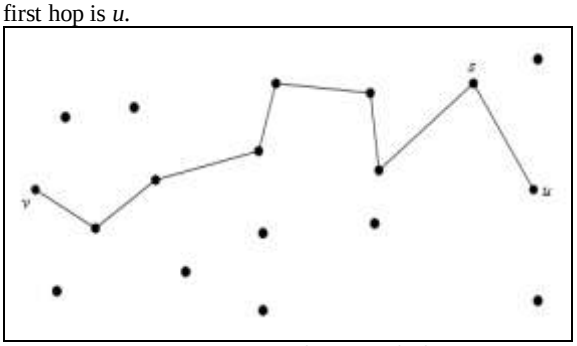

Figure 1: An illustration of the proof of Lemma 3, case 1. $\delta(u, v) > \delta(s, v)$

Case 3. If $\delta(u, v) = \delta(s, v)$, neither the shortest path from u to v passes through s , nor the shortest path from s to v passes through u; otherwise they could not have been of equal length. As we see that in case $\delta(u, v) = \delta(s, v) + 1$, in case 2, we have $\delta(u, v) = \delta(s, v) - 1$, and in case 3, we have $\delta(u, v) = \delta(s, v)$, and since all the above three cases are exhaustive, $\delta(u, v)$ can only have the value from $\delta(s, v) - 1$ to $\delta(s, v) + 1$. Thus we have:

$$\delta(s, v) - 1 \le \delta(u, v) \le \delta(s, v) + 1$$

as $u \in Q$, $\delta'(u, v) = \delta(s, v) + 1$. Substituting $\delta'(u, v) - 1$ for $\delta(s, v)$ we get:

$$\delta'(u, v) - 2 \le \delta(u, v) \le \delta'(u, v)$$

After rearranging, we get:

$$\delta(u, v) \le \delta'(u, v) \le \delta'(u, v) + 2$$

It implies that for every vertex $u \in Q$ the distance reported from every other vertex $v \in V/P$ has one sided additive error of at most 2.

Lemma 4. For every $u, v \in R$, there exists $\delta(u, v) \le \delta'(u, v) \le \delta'(u, v) + 2$.

Proof: For any two vertices $u, v \in R$, we can have following three mutually exclusive cases:

1. Every vertex in a shortest path from u to v is in R. In this case we have, $\delta(u, v) = \delta''(u, v)$. Also we will have, for every $s \in S$, $\delta''(u, v) \le \delta(s, u) + \delta(s, v)$, and thus
   $\delta'(u, v) = \min\{\delta'(u, v), \min_{s \in S}\{\delta(s, u) + \delta(s, v)\}\}$
   $= \delta''(u, v) = \delta(u, v)$
2. A shortest path from u to v passes through one of the special vertex $s \in P$. Thus, by the optimal substructure property of the shortest path, that shortest path from u to v constitute of a shortest path from u to s and a shortest path from s to v. Thus, we have, $\delta(u, v) = \delta(s, u) + \delta(s, v)$. As the shortest path passes through s, we will have for every $s' \in S$,
   $\delta(s, u) + \delta(s, v) \le \delta(s', u) + \delta(s', v)$,
   and
   $\delta(s, u) + \delta(s, v) \le \delta''(u, v)$.
   Thus,
   $\delta'(u, v) = \min\{\delta''(u, v), \min_{s \in S}\{\delta(s, u) + \delta(s, v)\}\}$
   $= \delta(s, u) + \delta(s, v)$
   $= \delta(u, v)$
3. No shortest path between u and v passes through any vertex $s \in P$.

Since we have excluded previous cases, a shortest path between u and v must pass through a vertex $w \in Q$. By the optimal substructure property of the shortest path, that shortest path from u to v constitutes of a shortest path from u to w and a shortest path from w to v. Thus we have, $\delta(u, v) = \delta(u, w) + \delta(w, v)$. Suppose $s \in P$ be the neighbor of w. Since the shortest path from u to v does not pass through s, the shortest path from u to w also does not pass through s, as later is a sub graph of former, as shown in figure 2.

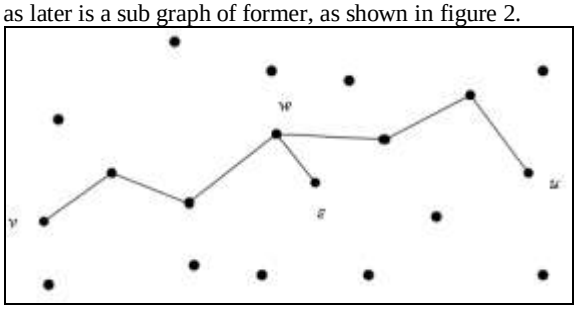

Figure 2: An illustration of the proof of Lemma 4, case 3. $\delta(u, v) = \delta(u, w) + \delta(w, v)$

Consider the path between u and w which consists of the shortest path between u and s of length $\delta(u, s)$ and the edge (w, s). The length of this path is $\delta(u, s) + 1$. As no shortest path between u and w passes through s, this path cannot be equal to or shorter than the shortest path between u and w. Thus, $\delta(u, s) + 1 > \delta(u, w)$ or $\delta(u, s) \geq \delta(u, w)$.

Now consider the path between u and s which consists of the shortest path between u and w of length $\delta(u, w)$ and the edge (w, s). The length of this path is $\delta(u, w) + 1$. Since no path between u and s can be shorter than the shortest path between u and s, $\delta(u, w) + 1 \geq \delta(u, s)$. As $\delta(u, s) \geq \delta(u, w)$, we have

$$\delta(u, w) \leq \delta(u, s) \leq \delta(u, w) + 1$$

By interchanging the roles of u and v, we get:

$$\delta(v, w) \leq \delta(v, s) \leq \delta(v, w) + 1$$

Adding the two inequalities, we get:

$$\delta(u, w) + \delta(v, w) \leq \delta(u, s) + \delta(v, s) \leq \delta(u, w) + \delta(v, w) + 2$$

Since $\delta(u, v) = \delta(u, w) + \delta(v, w)$, we get

$$\delta(u, v) \leq \delta(u, s) + \delta(v, s) \leq \delta(u, v) + 2$$

In this case for every $s' \in S$, we will have one of the following two cases:

(a) $\delta''(u, v) \leq \delta(s', u) + \delta(s', v)$. Thus we will have $\delta'(u, v) = \delta''(u, v)$. Since $\delta(u, v) \leq \delta(u, s) + \delta(v, s) \leq \delta(u, v) + 2$ and $\delta''(u, v) \leq \delta(s', u) + \delta(s', v)$, we have :
$\delta(u, v) \leq \delta'(u, v) \leq \delta(u, v) + 2$

(b) There exists s such that $\delta''(u, v) > \delta(u, s) + \delta(v, s)$ and $\delta(u, s) + \delta(v, s) \leq \delta(s', u) + \delta(s', v)$. Thus we will have
$\delta'(u, v) = \delta(u, s) + \delta(v, s)$, and hence,
$\delta(u, v) \leq \delta'(u, v) \leq \delta(u, v) + 2$

Thus it follows that if the shortest path does not pass through any of the special vertices, but through the neighbor of a special vertex, then

$$\delta(u, v) \leq \delta'(u, v) \leq \delta(u, v) + 2.$$

All the above three cases are exclusive and exhaustive. For case 1 and case 2, we have $\delta'(u, v) = \delta(u, v)$, and for case 3, we have $\delta(u, v) \leq \delta'(u, v) \leq \delta(u, v) + 2$.

Combining all the three cases we get:

$$\delta(u, v) \leq \delta'(u, v) \leq \delta(u, v) + 2.$$

It implies that for every vertex $u, v \in R$, the distance reported between u and v has one sided additive error of at most 2.

5. THE ALGORITHMS

Now we present an algorithm APASP based on our approach to find the optimized shortest paths between all pairs of vertices of a graph with one sided additive error of at most 2. Algorithm APSAP has four steps:

1. Identify special vertices and constructs sets P, Q and R.
2. Carry out BFS for each of the special vertices in the input graph.
3. Compute single source optimized shortest paths for each vertex in Q.
4. Compute single source optimized shortest paths for each vertex in R.

Algorithms for the subroutine to carry out these tasks are given in the following sections. Input to the algorithm APASP is an unweighted and undirected graph $G = (V, E)$. It calls the four subroutines sequentially and outputs an $n \times n$ matrix $dist[][]$; $dist[u][v]$ is the optimized shortest distance between vertex u and vertex v for every $u, v \in V$. Before calling the subroutines, $dist[][]$ is initialized such that $dist[u][v] = 1$ if $(u, v) \in E$, $dist[u][v]$ otherwise. If the input graph $G = (V, E)$ is connected, $dist[u][v]$ is initialized to $n + 1$, if(u, v) does not belongs to E.

Algorithm to CONSTRUCT *P, Q, R*

This algorithm identify the special vertices in $G = (V, E)$, and construct the sets P, Q, R. For every $u \in Q$, it also stores a vertex $s \in P$ in $neighbor[u]$, such that $(u, s) \in E$. Input to this algorithm is the graph $G = (V, E)$.

Algorithm 1: CONSTRUCT

Initialize: $G' = (V', E')$ and $G = (V, E)$ are equivalent.
$n = |V|$
$P = Q = ø$
while $|E'| > n\sqrt{n}$ do
Select a vertex s of maximum degree from V'.
    $P = Pus$
for every $u \in V$ such that $(u, s) \in E'$ do
    $Q = Quu$
    $neighbor[u] = s$
for every $v \in V$ such that $(u, v) \in E'$ do
    $E' = E' - (u, v)$
end for
    $E' = E' - (u, s)$
    $V' = V' - u$
end for
    $V' = V' - s$
end while
$R = V'$

Lemma 5. The running time taken by the algorithm CONSTRUCT is $O(m)$ if $m > n\sqrt{n}$, and $O(1)$ otherwise.

Proof: If $m \leq n\sqrt{n}$, then body of *while* loop will not be executed and hence time taken by algorithm is $O(1)$. Also if $m \leq n\sqrt{n}$, we will have $P = Q = ø$ and $R = V$. If $m > n\sqrt{n}$, the *while* loop will be iterated for $|S|$ times, since one vertex is chosen as special vertex in each iteration of *while* loop. Selecting the maximum degree vertex and updating the degree of adjacent vertices after deleting the maximum degree vertex will take $O(d)$ time, where d is the degree of the maximum degree vertex, which cannot be greater than n. In each iteration of *while* loop, the outer *for* loop is also

executed, and in each iteration of outer *for* loop, inner *for* loop is also executed. In each iteration of inner *for* loop an edge is deleted from the graph. As no edge is added to graph at any point of time, the number of statements executed will be of order of number of edges deleted, which cannot be more than *m*. Thus the time taken by algorithm, if $m > n\sqrt{n}$, will be $O(n|s| + m)$, which is $O(m)$ (since $|S| \leq \sqrt{n}$ and $m > n\sqrt{n}$).

Algorithm: Single Source Shortest Path$_P$ (SSSP$_P$)

This algorithm carries out BFS for every special vertex in $G = (V, E)$ and stores the shortest distances of every vertex from the special vertices in array *dist*[][]. Input to this algorithm is the graph $G = (V, E)$ and set of special vertices *P*.

Algorithm 2: SSSP$_P$

```
for every s ∈ P do
        run BFS in G = (V, E) with s as root.
          for every v ∈V do
                dist[s][v]  = dist[v][s] = δ(s, v)
                (computed during BFS)
          end for
end for
```

Lemma 6. If $m > n\sqrt{n}$ the running time taken by the algorithms SSSPP is $O(m\sqrt{n})$.

Proof: As SSSPP runs the BSF for every special vertex in $G = (V, E)$, the running time would be $O(m|S|)$. Since $|S| \leq \sqrt{n}$ if $m > n\sqrt{n}$, algorithm SSSPP will take $O(m\sqrt{n})$ running time.

Algorithm: Single Source Optimized Shortest Path$_Q$ (SSOSP$_Q$)

This algorithm computes the optimized shortest distance of the neighbors of the special vertices from every other non-special vertex and stores the optimized shortest distance in array *dist*[][]. Input to this algorithm is the graph $G = (V, E)$, *P*, *Q*, *neighbor*[].

Lemma 7. If $m > n\sqrt{n}$ the running time taken by the algorithm SSOSPQ is $O(n^2)$.

Algorithm 3: SSOSP$_Q$

```
for every u ∈ Q do
    for every v ∈ V/P do
          if dist[u][v] > dist[neighbor[u]][v]+1 then
            dist[u][v] = dist[v][u] = dist[neighbor[u]][v] + 1
          end if
    end for
end for
```

Proof: As SSOSP$_Q$ consists of two nested *for* loops, the running time would be $O(|Q||V|)$, because outer *for* loop is iterated for every vertex in *Q* and the inner *for* loop is iterated for every vertex in *V/P*. If $m < n\sqrt{n}$, then $|Q| = 0$ else $|Q|$ can not be greater than *n*. Hence algorithm SSOSP$_Q$ will take $O(n^2)$ running time, If $m > n\sqrt{n}$

Algorithm: Single Source Optimized Shortest Path$_R$ (SSOSP$_R$)

This algorithm computes the optimized shortest distance between all pairs of vertices in *R* and stores the optimized shortest distance in array *dist*[][]. Input to this algorithm is *P*, *R* and $G' = (V', E')$, the subgraph of $G = (V, E)$ induced by the vertices in *R*. Subgraph $G' = (V', E')$ is obtained from the algorithm CONSTRUCT.

Lemma 8. The running time taken by the algorithm SSOSPR is $O(n^{5/2})$ if $m > n\sqrt{n}$, $O(mn)$ otherwise.

Proof: Algorithm SSOSP$_R$ runs the BFS in $G' = (V', E')$, for every vertex in *R*, which will take $O(|R||E'|)$ time. Then it executes three nested *for* loops, which will take $O(|R|^2|P|)$, because two outermost *for* loop are iterated for every vertex in *R* and the inner *for* loop is iterated for every vertex in *P*. Thus, the total time taken by algorithm SSASP$_R$ is $O(|R||E'|+ |R|^2|P|)$. If $m > n\sqrt{n}$, it would become $O(n^{5/2})$ as in this case, $|R| = O(n)$, $|E'| < n\sqrt{n}$ and $|P| < \sqrt{n}$. If $m < n\sqrt{n}$, it would become $O(mn)$, as we have, $R = V$, $E' = E$ and $P = ø$. Thus, algorithm SSOSP$_R$ takes $O(n^{5/2})$ running time if $m > n\sqrt{n}$, $O(mn)$ otherwise.

Algorithm 4: SSOSP$_R$

```
for every u ∈ R do
          run BFS in G' = (V', E') with u  as root.
        for every v ∈ R do
                  dist[u][v] = dist[v][u] = δ(u, v)
                  (computed during BFS)
          end for
end for
for every u ∈ R do
   for every v ∈ R do
      for every s ∈ S do
          if dist[u][v] > dist[s][u] + dist[s][v] then
             dist[u][v] = dist[v][u] = dist[s][u] + dist[s][v]
          end if
      end for
   end for
end for
```

Analysis of APOSP

Lemma 9. Algorithm APOSP runs in $O(\min(mn, n^{5/2}))$ time.

Proof: From Lemmas, we have if $m > n\sqrt{n}$, the running time of algorithm APOSP will be $O(m + m\sqrt{n} + n^2 + n^{5/2})$ i.e., $O(n^{5/2})$ else it will be $O(mn)$. Since $n^{5/2} < mn$ when $m > n\sqrt{n}$, the running time taken by algorithm APOSP is $O(\min(mn, n^{5/2}))$.

Lemma 10. Algorithm APOSP reports the shortest distance between every pair of vertices with one sided additive error of at most 2.

Proof: A vertex belongs to *P*, *Q* or *R*. Thus, for any pair of vertices, if one of the vertices is in *P*, the true shortest distance is reported (since BFS is run on the input graph for every vertex of *P*). If neither of the two vertices is in *P*, but one of the vertex (or both) is in -*Q*, the shortest distance reported has one sided additive error of at most 2. And if both vertices are in *R*, even then the shortest distance reported has one sided additive error of at most 2. Thus for every pair of vertices algorithm APOSP reports the shortest distance with one sided additive error of at most 2. Algorithm APOSP can be easily modified to report one of the optimized shortest paths between every pair of vertices, without increasing the time complexity by more than the size of the output. Suppose the shortest path between two vertices *u* and *v* is to be reported. If one of the vertex, lets say *u*, is in *P*, BFS in $G = (V, E)$ for *u* returns the breadth-first tree rooted at *u*, which contains the shortest paths form *u* to every other vertex. If $u \in Q$ and $s \in P$ is a neighbor of *u*, then report the path between *u* and *v* in the breadth-first tree rooted at *s*. If $u, v \in R$ and $s' \in P$

such that for every $s' \in P$ $\delta(s', u) + \delta(s', v) \leq \delta(s, u) + \delta(s, v)$, then report the path between u and v in the breadth-first tree rooted at s' or the path between u and v in the breadth-first tree rooted at u or v (returned by the BFS for u or v in G'), whichever is minimum.

All Pairs Shortest Paths and Maximal Independent Sets

Now we explore the problem of finding a quadratic running time algorithm for reporting all pair's optimized shortest paths with one sided additive error of at most 2. We will try to identify some graph for which it is possible. First we will find a suitable maximal independent set and then use this information to compute the all pairs approximate shortest paths.

All pairs approximate shortest paths algorithm using maximal Independent Set

If we are given a maximal independent set, then computation of all pairs approximate shortest paths, with one sided additive error of at most 2, can be done in $O(ms)$ time, where s is the size of maximal independent set. Let $G = (V, E)$ be an unweighted and undirected graph .Suppose the set of vertices M (V forms a maximal independent set). For Computing the approximate shortest path we will first run BFS on $G = (V, E)$, for each vertex in M. Thus for each $m \in M$, we have the exact shortest distance from every other vertex in V. Since M is a Maximal Independent set, every vertex $u \in V/M$ has a neighbor in M. Let u and $m \in M$ be one of its neighbors. If we have to compute the shortest distance of u from $v \in V/M$, then we will report $\delta(m, v) + 1$ as the optimized shortest distance. It can be easily seen that $\delta(u, v) \leq \delta(m, v) + 1 \leq \delta(u, v) + 2$. Thus for every $u \in V/M$, we can now find the optimized shortest distance of u from any vertex in V, with one sided additive error of at most 2, in constant time, if we have stored a neighbor $m \in M$ of u. Thus the time taken by algorithm is $O(m|M|)$.

Finding maximal independent set of small size

We now study finding upper bounds on the size of maximal independent set of some particular type of graphs. Consider the following heuristic to construct a maximal independent set M of a connected graph $G = (V, E)$.

1. Choose a vertex v of degree d; d greater than or equal to the average degree of the graph, such that the number of edges removed from the graph after deleting the vertex v and its neighbors is $O(m^2/n^2)$.
2. Delete vertex v and all of its neighbors, including the edges incident on these vertices, from the graph.
3. Repeat steps 1-2 for residual graph until all the vertices are deleted.

Lemma 11. The size of maximal independent set M is $O(n^2/m)$.

Proof: We have to prove that $|M| \leq cn^2/m$. The proof is by induction on n.
Let the value of constant c be 1. Lemma is clearly true when $n = 2$ and $m = 1$. Assume that the lemma is true for all graphs having less than n vertices.

Let the vertex v have degree d; d greater than or equal to the average degree i.e., $2m/n$ of the graph, such that the number of edges removed δ from the graph after deleting the vertex v and its neighbors is $O(m^2/n^2)$. Delete vertex v and all d of its neighbors. Thus residual graph has $n - d - 1$ vertices and $m - \delta$ edges.

Inductively residual graph will have a maximal independent set of size at most $(n - d - 1)^2/(m - \delta)$. Thus original graph has a maximal independent set M of size at most n^2/m, if

$$n^2/m \geq 1 + (n - d - 1)^2/(m - \delta)$$

Let $\alpha = d + 1$. Since $n > d \geq (2m/n)$, $n \geq \alpha > (2m/n)$. Thus, $|M| \leq n^2/m$, if

$$(n^2/m) \geq 1 + (n - \alpha)^2/(m - \delta)$$

or if,

$$mn^2 - \delta n^2 \geq m^2 - \delta m + mn^2 + \alpha^2 m - 2\alpha mn$$

or if,

$$m^2 + \delta(n^2 - m) + \alpha^2 m - 2\alpha mn \leq 0$$

$\alpha^2 m - 2\alpha mn$ is minimum at $\alpha = n$. Since $n \geq \alpha > (2m/n)$ at $\alpha = (2m/n) + 1$, $\alpha^2 m - 2\alpha mn$ will have its maximum value, which will be less than $4m^3/n^2 - 4m^2$ (obtained after substituting $\alpha = 2m/n$ in $\alpha^2 m - 2\alpha mn$). Thus, $m^2 + \delta(n^2 - m) + \alpha^2 m - 2\alpha mn \leq m^2 + \delta(n^2 - m) + 4m^3/n^2 - 4m^2$.

Thus $|M| \leq n^2/m$, if $m^2 + \delta(n^2 - m) + 4m^3/n^2 - 4m^2 \leq 0$, or if, $\delta \leq (3m^2 - (4m^3/n^2))/n^2 - m$, which is true. Hence size of maximal independent set M is at most n^2/m.

Clearly the maximal independent set M can be found in $O(n^2)$ time using a greedy algorithm. For each vertex $v \in V$, we will add the degree of its neighbors. Let $\gamma(v)$ is the sum of degree of all neighbors of v and $\delta(v)$ is the number of edges removed from the graph, if vertex v and its neighbor are deleted from the graph. It is clear that $\delta(v) \leq \gamma(v) \leq 2\delta(v)$, since in the sum of degree of neighbors, an edge that is going to be deleted might be counted once or it may be counted twice (if both vertices, on which the edge is incident, are the neighbor of v). Also, $\delta(v) = \gamma(v)$, if and only if no two neighbors of v are adjacent. Computing $\gamma(v)$, will take $O(d_v)$ time, where d_v is the degree of vertex v. Thus computing $\gamma(v)$ for all the vertices of the graph will take $O(\sum_{v \in V} d_v) = O(m)$ time. To update degree d_v and $\gamma(v)$ for every vertex $v \in V$, we subtract 1 and d_u from d_v and $\gamma(v)$, respectively, whenever an edge $(u, v) \in E$ is deleted form the graph. Since there are m edges in graph and no edge is added in the graph while constructing M, the total time taken to update d_v and $\gamma(v)$, for every vertex $v \in V$ will be $O(m)$. To identify a vertex in M will take $O(n)$ time, if we do a linear search on d_v and $\gamma(v)$, for every vertex $v \in V$. The size of M is $O(n^2/m)$. Thus the total time required to construct M is $O(m + n.n^2/m)$ i.e., $O(m + n^3/m) = O(n^2)$.

Sub-cubic running time algorithm for computing all pairs shortest paths

Suppose we are given an independent set of vertices (not necessarily maximal) of sub linear size. If the number of edges in the residual graph, after deleting the vertices in the independent set, is sub-quadratic, then the shortest paths between all pairs of vertices in graph can be computed in sub-cubic time. For this, we will compute the single source shortest paths in input graph for each vertex in independent set. Then for remaining vertices, we will compute the single source shortest paths in residual graph (obtained after deleting vertices in independent set). Shortest path between every two vertices u and v in the independent set will be either in residual graph or through one or more vertices in independent set, which can be calculated as $\min_w\{\delta(w, u) + \delta(w, v)\}$ for every vertex w in independent set. Since size of independent set is sub-linear

and number of edges in the residual graph is sub-quadratic, this all can be done in $O(n^3)$ time.

6. CONCLUSIONS

In this paper we described an $O(n^{5/2})$ running time algorithm to compute all pairs optimized shortest paths in an unweighted and undirected graph, with one sided additive error of at most 2. We also studied the problem of finding quadratic running time algorithm for some particular types of graphs and tried to identify them. We also studied upper bounds on the size of a maximal independent set of such graphs.

Our technique can be enhanced to find the optimized shortest paths in a weighted graph. For a directed graph however, this technique is not likely to work, since in a directed graph the adjacency relation is not commutative and thus a path *p* from *u* to *v* is not a path from *v* to *u*. The commutative property of the adjacency relation is the key property of undirected graph, which we utilize in our technique.

7. REFERENCES

[1] Aingworth, D., Chekuri C., Indyk P., Motwani R., "Fast estimation of diameter and shortest paths (without matrix multiplication)", SIAM Journal on Computing 28 (1999), 1167-1181.

[2] Alon N., Galil Z., Margalit O., "On the exponent of the all pairs shortest path problem", Journal of Computer and System Sciences 54 (1997) 255 -262.

[3] Alon N., Naor M. Derandomization, "Witnesses for Boolean matrix multiplication and construction of perfect hast functions", Algorithms 16 (1996), 434-449.

[4] Baswana S., Goyal V., Sen S., "All-pairs nearly 2-approximate shortest-paths in $o(n^2$ polylog $n)$ time", In STACS 2005: 22nd Annual Symposium on Theoretical Aspects of CS (2005), pp. 666- 679.

[5] Chvtal V., "A greedy heuristic for the set-covering problem", Math. Oper. Res. 4 (1979), 233-235.

[6] Cohen E., Zwick U., "All-pairs small-stretch paths", Journal of Algorithms 38 (2001), 335- 353.

[7] Coppersmtts D., Winoged S., "Matrix multiplication via arithmetic progression", Journal of Symbolic Computation 9 (1990), 251-280.

[8] Cormen T., Stein C., Rivest R., Leiserson C., "Introduction to Algorithms", PHI, 2001.

[9] Dijkstra, E., "A note on two problems in connection with graphs. Numeric Mathematic 1 (1959), 269-271.

[10] Dor D., Halperin S., Zwick U., "All pairs almost shortest paths", SIAM Journal on Computing 29 (2000), 1740-1759.

[11] Fredmian M., "New bounds on the complexity of the shortest path problem", SIAM Journal of computing 5 (1976), 83- 89.

[12] Galil Z., Margalit O., "Witnesses for Boolean matrix multiplication", Journal of Complexity 9 (1993) 201- 221.

[13] Galil Z., Margalit O., "All pairs shortest paths for graph with small integer length edge", Journal of Computer and System Science 54 (1997) 243-254.

[14] Hagerup T., "Improved shortest paths on the word ram", In Proceedings of the 27th International Colloquium on Automata, Language and Programming (2000),pp. 61 -72.

[15] Halldorsson M., "Approximating the minimum maximal independence number", Inf. Process. Lett. 46, 4 (1993), 169-172.

[16] Johnson D., "Efficient algorithms for shortest paths in sparse graphs", Journal of the ACM 24 (1977), 1-13.

[17] Karger D., Koller D., Phillips S., "Finding the hidden path: time bounds for all-pairs shortest paths", SIAM Journal on Computing 22 (1993), 1199 -1217.

[18] Lovasz L., "On the ratio of optimal integral and fractional covers", Discrete Math. 13 (1975), 383 - 390.

[19] Mcgeoch G., "All-pairs shortest paths and the essential sub graph", Algorithmic 13 (1995), 426-461.

[20] Pettie S., Ramachandran V., "Computing shortest paths with comparisons and additions", In SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (2002), pp. 267- 276.

[21] Seidel R., "On the all-pairs-shortest-path problem in unweighted undirected graphs", Journal of Computer and System Science 51 (1995), 400- 403.

[22] Takaoka T., "A new upper bound on the complexity of the all pairs shortest path problem", Information Processing Letters 43 (1992), 195-199.

[23] Thorup M., "Undirected single-source shortest paths with positive integer weights in linear time", Journal of the ACM 46 (1999), 362- 394.

[24] Thorup M., "Floats, integers, and single source shortest paths", Journal of Algorithms 35 (2000), 189 -201.