All-Pair Shortest Path

In any finite set, relationships between elements can be represented as a graph, where vertices are elements and edges denote connections, such as distance, cost, or any other form of values. One of the most interesting problems is the shortest path problem: Single Source Shortest Path (SSSP) and All Pairs Shortest Path (APSP), which find the shortest routes from one or all vertices. SSSP and APSP are interesting because of the polynomial time.

In 1959, Dijkstra published an algorithm to find the SSSP that runs in $O(V^2)$, using the idea of edge relaxation he had devised three years earlier (Dijkstra, 1959). Bellman, Ford, and Moore independently discovered the algorithm to search for SSSP in $O(VE)$ time in 1955, 1958, and 1959, respectively (Bang-Jensen & Gutin, 2008). Dijkstra's algorithm works under the BFS and greedy principles, which runs in $O(V+E)$ (assuming all operations are $O(1)$) (Cormen et al., 2022). The Bellman-Ford-Moore (BFM) algorithm loops through $V-1$ iterations, and in each iteration, it relaxes all edges E in the graph (Cormen et al., 2022). Being a brute-force approach, the latter works on both negative and positive weights. The former, due to its greedy nature, cannot handle negative weights. Both can work with cycles, but negative cycles, especially for BFM's, present undefined results(Cormen et al., 2022).

A list is not an efficient data structure, especially in retrieving minimum values. The two key operations in Dijkstra's algorithm are retrieving the unvisited vertex with the minimum distance from the source and decreasing the key value (update) in the edge relaxation step. At the time, Dijkstra described the algorithm as using three sets, the set that contains the edges is effectively a list, the algorithm has to check the whole list for the minimum unvisited vertex, but to decrease-key, it can access the list in $O(1)$, making the whole algorithm $O(V^2+E)$ or $O(V^2)$ (Richards, n.d.; Dijkstra, 1959). Johnson (1977) proposed the binary heap, with all operations being $O(\log n)$, lowering the complexity to $O((V+E)\log V)$. Fredman and Tarjan (1984) took it a step further with the Fibonacci heap to have only the delete operation be $O(\log n)$ and all the others be $O(1)$, which results in $O(V\log V+E)$.

It is necessary to introduce the principles of these two algorithms because most later research on both SSSP and APSP is done based on them. To know the shortest path from every vertex to all others using these SSSP algorithms, it is simply to run them V times, which is where the problem is. At best, Dijkstra's and BFM's algorithms run in $O(V\log V+E)$ and $O(VE)$, respectively. A graph is an interesting data structure because the number of edges E can vary. E is in the same order as V if the graph is sparse, but it can be V^2 if the graph is dense; thus, the complexities can be $O(V \log V + V^2)$ and $O(V^3)$. Running each of these V times makes it asymptotically $O(V^3)$ and $O(V^4)$.

Floyd and Warshall independently discovered the brute-force algorithm to perform APSP in 1962 (Warshall, 1962; Floyd, 1962). The algorithm works on the idea that it will compute the shortest distances between two vertices by introducing intermediate vertices. The number of intermediate vertices k goes from 1 and up to n. There are three nested loops. The first one is the k intermediate vertices, the second one is the source i, and the third one is the destination j, all of which iterate from 1 to n. At each k, the matrix from $k-1$ is the shortest distance between each pair of source-destination vertices with $k-1$ intermediate vertices. The matrix is updated at each (k, i, j) to get the minimum value between the current distance of the source-destination $weight(i, j)$ and the distance with k intermediate vertices $(\text{weight}(i, k) + \text{weight}(k, j))$; this results in the consistent complexity of $O(V^3)$.

In his 1977 paper, Johnson also introduced an algorithm for APSP based on Dijkstra's (Johnson, 1977). One drawback of Dijkstra's algorithm is that it cannot handle negative-weight edges(Cormen et al., 2022). Johnson overcame this by combining both BFM's algorithm and Dijkstra's (Fibonacci Heap) algorithm, reweighting all edges to be positive. First, it adds a vertex s, where the distance from s to other vertices is initially $0, O(V + E)$. Then, it performs BFM's algorithm to get the shortest distances from s and to check for negative cycles, $O(VE)$; it uses the shortest distances from s to adjust the weights, making them non-negative, $O(E)$. Finally, it performs Dijkstra's algorithm on the updated-weight graph for all vertices, $O(V^2 \log V + VE)$. Together, the complexity is asymptotically $O(V^2 \log V + VE)$. When the graph is dense, it is effectively $O(V^2 \log V + V^3)$, making it in the same order as Floyd-Warshall's. However, in most real-life cases, the graph is sparse, meaning that Johnson's is still better than Floyd-Warshall's (Johnson, 1977; Cormen et al., 2022).

Other algorithms have explored different methods based on these two approaches. In 1976, Fredman (1976) proposed a theoretical matrix multiplication technique to perform APSP on non-negative edges in $O(V^{\frac{5}{2}})$. Pettie (2004) introduced a theoretical model based on component hierarchy proposed by Thorup (2004), which lowers complexity to $O(V^2 \log \log V + VE)$. Chan (2008) proposed another theoretical algorithm, an improvement on previous attempts such as Fredman's (1976), to perform APSP in $O(\frac{V^3}{\log V})$.

In conclusion, for a graph G with V vertices and E edges, Dijkstra's algorithm performs in $O(V \log(V) + E)$, BFM's in $O(VE)$ for SSSP. Regardless of the connectivity, Dijkstra's outperforms BFM's. For APSP, Floyd-Warshall works in a consistent complexity of $O(V^3)$; Johnson's algorithm fixes Dijkstra's negative-weight problem by reweighting all edges to be positive, running in $O(V^2 \log(V) + VE)$. For a sparse graph, it is worth running Johnson's($O(V^2 \log(V))$), but for a dense one, both algorithms run in the order of $O(V^3)$. All of these algorithms can run on negative-weight edges(Dijkstra's needs to be run using Johnson's reweighting), but not negative-weight cycles. Many other attempts have been made besides these, yet they can only optimize it to sub-cubic complexity like Fredman (1976) or Pettie (2004).

References

1. Warshall, S. (1962). A theorem on Boolean matrices. *Journal of the ACM, 9*(1), 11–12. https://doi.org/10.1145/321105.321107
2. Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM, 5*(6), 345. https://doi.org/10.1145/367766.368168
3. Bang-Jensen, J., & Gutin, G. Z. (2008). Digraphs: Theory, algorithms and applications (2nd ed.). Springer. https://doi.org/10.1007/978-1-84800-998-1
4. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271. https://doi.org/10.1007/BF01386390
5. Fredman, M. L. (1976). New Bounds on the Complexity of the Shortest Path Problem. *SIAM J. Comput., 5*(1), 83–89. https://doi.org/10.1137/0205006
6. Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, *24*(1), 1–13. https://doi.org/10.1145/321992.321993
7. Thorup, M. (2004). Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences*, *69*, 3. https://doi.org/10.1016/j.jcss.2004.04.003
8. Pettie, S. (2004). A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, *312*, 1. https://doi.org/10.1016/S0304-3975(03)00402-X
9. Chan, T. M. (2008). AllPairs Shortest Paths with Real Weights in O(n3/log n) Time. *Algorithmica*, *50*(2), 236–243. https://doi.org/10.1007/s0045300790621
10. Richards, H. (n.d.). *Edsger W. Dijkstra - A.M. Turing Award Laureate.* Association for Computing Machinery. Retrieved February 5, 2025, from https://amturing.acm.org/award_winners/dijkstra_1053701.cfm
11. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Single-source shortest paths. In *Introduction to algorithms* (4th ed., chapter 22). The MIT Press.
12. Fredman, M. L., & Tarjan, R. E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, 338–346. https://doi.org/10.1109/SFCS.1984.715934