iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH
IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: XII Month of publication: December 2025

DOI: https://doi.org/10.22214/ijraset.2025.75934

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

A Review and Comparative Analysis of Foundational Shortest-Path Algorithms

Er. Harjasdeep Singh[1], Suneha Gumber[2], Rishu[3]
[1]Assistant Professor, [2, 3]Student, Dept. of CSE MIMIT, Malout

***Abstract:** Theshortest-pathproblem,afundamentalchallenge ingraphtheoryandcomputerscience,seekstofindapath of minimum cumulative weight between vertices in a weighted graph. Its solution is critical to a vast array of applications, includingnetworkrouting,logistics,robotics,andbioinformatics. This paper provides a comprehensive review and comparative analysis of four foundational algorithms that address this prob- lem: Dijkstra's algorithm, the Bellman-Ford algorithm, the A* search algorithm, and the Floyd-Warshall algorithm. We delve into the historical context, theoretical underpinnings, and oper- ational mechanics of each method. The analysis contrasts these algorithms across several key dimensions: algorithmic paradigm (greedy,dynamicprogramming,heuristicsearch),problemscope (single-source vs. all-pairs), handling of edge weights (including negative values), and computational complexity. By juxtaposing their respective strengths, weaknesses, and ideal use cases, this review illustrates that the selection of an optimal shortest-path algorithm is not a matter of absolute superiority but a nuanced decision contingent upon specific problem constraints such as graph structure, edge weight properties, and the required scope of the solution. The paper concludes by contextualizing these classic algorithms as essential building blocks for modern, more complex pathfinding solutions and highlights ongoing research that continues to refine our understanding of this classic compu- tational problem.*

***Index Terms:** Graph Theory, Shortest Path, Dijkstra, Bellman-Ford, A* Search, Floyd-Warshall, Algorithm Analysis, Computational Complexity.*

I. INTRODUCTION

Theproblemoffindingtheshortestpathinanetworkisone of the most studied problems in computer science, operations research, and graph theory [1], [2]. In its most general form, theprobleminvolvesaweighted graph,G=(V,E), where V isasetofvertices(ornodes) and$E$isasetofedgesconnecting pairsofvertices. Eachedge(u, v)∈Eisassignedanumerical weight, w(u,v), which can represent cost, distance, time, or anyotherquantifiablemeasureoftraversingtheedge[3], [4].Theobjectiveistofindapathbetweentwospecified vertices—asourceand adestination—such that thesum of the weights of the edges constituting the path is minimized.

Theapplicationsofshortest-pathalgorithmsarepervasive andintegraltomoderntechnologyandinfrastructure.They form theback boneofdigitalmapping serviceslikeGoogle Mapsforrouteplanning,networkroutingprotocolssuchas OSPF (Open Shortest Path First) for directing internet traffic, resourceallocationinlogisticsandsupplychainmanagement, and pathfinding for autonomous agents in robotics and video games [4]–[6]. The versatility of the graph model allows ver- tices to represent not just physical locations but also abstract states, with edges representing transitions, making shortest- path algorithms a powerful tool for solving a wide range of optimization problems [3].

Theliteraturepresentsarichtapestryofalgorithmsdesigned to solve this problem, each tailored to different constraintsand problem variations [1], [2]. These variations are broadly classified into two categories: the Single-Source Shortest Path (SSSP) problem, which aims to find the shortest paths from a single source vertex to all other vertices in the graph, and the All-Pairs Shortest Path (APSP) problem, which seeks to find the shortest path between every pair of vertices [1], [3].

This review focuses on four seminal algorithms that have become canonical in the study of this problem:

1) Dijkstra's Algorithm: A greedy algorithm that effi- ciently solves the SSSP problem for graphs with non- negative edge weights.
2) The Bellman-Ford Algorithm: A more robust SSSP algorithm based on dynamic programming, capable of handling graphs with negative edge weights and detect- ing negative-weight cycles.
3) The A* Search Algorithm: An informed or heuristic searchalgorithmthatextendsDijkstra'sapproachtofind the shortest path between a single pair of vertices more efficientlybyusingproblem-specificknowledgetoguide its search.
4) The Floyd-Warshall Algorithm: A dynamic program- mingalgorithmthatsolvesthe APSP problem,elegantly handlingbothpositiveandnegativeedge weights(inthe absence of negative-weight cycles).

The continued relevance of these distinct algorithms under- scores a critical reality: there is no single ”best” algorithm for allshortest-pathscenarios.Theoptimalchoiceiscontingenton the specific characteristics of the problem at hand, including thegraph’ssizeanddensity,thenatureofitsedgeweights,and whether thesolutionrequires paths fromasinglesourceorbe- tweenallpairsofnodes. Thispaperaimstoprovidea rigorous comparative analysis of these four foundational algorithms. It will examine their historical origins, formalize their opera- tional principles, analyze their computational complexity, and contrast their performance under various conditions. Through thisdetailed exploration, weseekto furnishaclear framework for understanding the trade-offs involved and for selecting the most appropriate algorithm for a given application.

II. LITERATURE REVIEW AND HISTORICAL CONTEXT

The development of shortest-path algorithms is deeply in- tertwined with the history of computer science, reflecting a fascinating evolution of computational paradigms. The four algorithms under review did not emerge in a simple linear progressionofimprovementbutratherrepresentdistinctphilo- sophical approaches to problem-solving: greedy optimization, dynamic programming, and heuristic search.

A. *Dijkstra’sAlgorithm:TheGreedyApproach*

In 1959, Dutch computer scientist Edsger W. Dijkstra pub- lished ”ANote on Two Problems in Connexion with Graphs,” which introduced his now-famous algorithm for the SSSP problem [7]. The algorithm was conceived in just 20 minutes in 1956 as a demonstration of the capabilities of the ARMAC computer [1], [2]. Its design embodies the greedy paradigm:at each step, it makes the locally optimal choice by selecting the unvisited vertex closest to the source, with the assumption thatthischoicewillleadtoagloballyoptimalsolution[3], [4]. This elegant and efficient approach established a benchmark for solving the SSSP problemon graphs with non- negative weights, and its core ”relaxation” process remains a fundamental concept in the field [5], [8].

B. *Bellman-Ford: ThePower ofDynamicProgramming*

The ability to handle negative edge weights, a crucial requirement in fields like economic modeling and certain network protocols, was addressed through the principle of dy- namic programming. The algorithmknown today as Bellman- Ford has a dual origin. Richard Bellman, in his 1958 paper ”On a Routing Problem,” developed a functional equation based on dynamic programming to solve routing issues [9]. Hismethoditeratively calculates theshortest path of atmost $k$edgesusingtheknownshortestpathsofat most$k-1$ edges [3], [4]. Independently, Lester R. Ford Jr., in his 1956 RAND Corporationreportonnetworkflowtheory,developedasimilar iterative method [10].This methodical, bottom-up approachis morecomputationallyintensivethanDijkstra’sgreedystrategy but offers greater versatility, including the critical ability to detect negative-weight cycles—a condition under which the shortest path is often undefined [5], [11].

C. *A*Search:TheHeuristicInfusion*

While Dijkstra’s and Bellman-Ford’s algorithms explore paths radiating from the source, the A* search algorithm introduced a sense of direction. Developed by Peter E. Hart, NilsJ. Nilsson, and BertramRaphaelat the Stanford Research oftenexploringamuchsmallerportionofthegraphthan its uninformed counterparts [5], [6]. This fusion of a formal graph search with domain-specific knowledge represented a significant step forward for goal-directed pathfinding.

D. *Floyd-Warshall:TheAll-PairsPerspective*

ThefocusshiftedfromsinglesourcetoallpairsshortestpathswiththealgorithmnowwidelyattributedtoRobertW.FloydandStephenWarshall. In1962,Floydpublished ”Algorithm97:ShortestPath,” whichpresentedaremarkably concisedynamicprogrammingsolutionforthe APSP problem [13]. Its recurrence relation is based on iterating through all possible intermediateverticesfor each pair of start and end nodes[3],[4].However,thealgorithm’sintellectuallineageis morecomplex.BernardRoyhadpublishedasimilaralgorithm in1959 intheFrenchjournal *ComptesRendusdel’Acade´mie desSciences*[14].Furthermore,StephenWarshall,alsoin 1962,publishedan algorithmforfindingthetransitiveclosure ofagraph(”AtheoremonBooleanmatrices”),whichis structurallyidenticalto Floyd’salgorithmwhenappliedto unweightedgraphs[4],[15].Thishistoryhighlightsacommon themeinscience: thesimultaneousand independentdiscovery of fundamental ideas. The Floyd-Warshall algorithm, as it is nowknown, providesapowerful,ifcomputationallydemand- ing, methodforcomprehensivepathanalysisindensegraphs.

These historical threads reveal that the field did not simply iterate towards a single ”best” algorithm. Instead, it branched out, exploring distinct computational philosophies to create a versatiletoolkit,witheachtooluniquelysuitedtoaspecific setofconstraintsandobjectives.

III. METHODOLOGICAL FRAMEWORK FOR ANALYSIS

Toconductarigorousandsystematiccomparisonofthefour selected algorithms,thispaperestablishesaformalframework grounded in graph theory and computational complexity anal- ysis.Thisframeworkdefinestheproblemdomainandoutlines the criteria against which each algorithm will be evaluated.

A. Graph Theory Preliminaries

Let$G$=(V,E)beadirectedgraph, where$V$isafinite setofvertices(nodes),$|V|=n$,and$E \subseteq V \times V$isasetof edges,$|E|=m$.Eachedge(u, v)∈Eisassociatedwitha real-valuedweight$w(u,v)$.Apath$p$fromavertex$v0$toa vertexvkisasequenceof vertices$\langle v0,v1,...,vk\rangle$ suchthat $(vi-1,vi)\in E$for all$1\leq i\leq k$.Theweightofapath is the sum of the weights of its constituent edges: $w(p)$ = Institute and published in their 1968 paper, ”A Formal Basis fortheHeuristicDeterminationofMinimumCostPaths,” A*$\sum ki=1w(vi-1,vi)$.was a product of the artificial intelligence community, specif- ically the Shakey the Robot project [6], [12]. A* enhances Dijkstra'sgreedyframeworkbyincorporatingaheuristicfunc-Theshortest-pathweight$\delta(u,v)$ fromvertex$u$to vertex$v$isdefinedas: (min{$w(p)$:$u^p v$}ifapathexists tion, $h(n)$, which estimates the cost from a given vertex n to the destination [4], [6]. Byprioritizing vertices that appear to beonthe mostpromisingpath to thegoal(i.e.,those withthe lowest$f(n)=g(n)+h(n)$), A*intelligentlyguidesitssearch, $\delta(u,v)=$ ∞ otherwise Ashortestpathfrom$u$to$v$isanypath$p$withweight$w(p)=\delta(u,v)$.

B. Problem Variants

This review addresses two primary variants of the shortest- path problem [1], [3]:

1) Single-Source Shortest Path (SSSP): Given a graph G andasourcevertex$s\in V$,findtheshortest-pathweight $\delta(s, v)$ for all $v \in V$. Dijkstra's,Bellman-Ford,and A* (for a single destination) address this problem.
2) All-Pairs Shortest Path(APSP): Findtheshortest-path weight$\delta(u,v)$foreverypairofvertices$(u,v)\in V\times V$. The Floyd-Warshall algorithm is designed for this problem.

C. Criteria for Comparative Analysis

Each algorithm will be evaluated based on the following criteria:

1) Algorithmic Paradigm: Theunderlyingdesignstrategy (e.g., greedy, dynamic programming, heuristic search).
2) Time Complexity: The asymptotic upper bound on the algorithm's running time, expressed in Big-O notationas a function of n and m.
3) Space Complexity: Theasymptotic upperboundonthe memory required by the algorithm.
4) Edge Weight Constraints: The algorithm's ability to function correctly with non-negative, negative, or zero- weight edges.
5) Negative Cycle Detection: The capability of the al- gorithm to detect the presence of a negative-weight cycle, a cyclewhose edges sum to a negative value.The shortest path is undefined in graphs containing a negative cycle reachable from the source, as traversing the cycle indefinitely would lead to an infinitely small path weight [3], [11].

D. The Role of Data Structures and Graph Density

The theoreticalcomplexity of analgorithmisanabstraction thatcanbesignificantlyinfluencedbyimplementationchoices, particularly the data structure used to represent the graph and the graph's density. A graphis considered **dense** if$m$is close to its maximum possible value, $O(n^2)$, and **sparse** if $m$ismuch smaller, often close to $O(n)$[4].

1) Adjacency Matrix: An n×nmatrix where the entryat (i,j)stores the weight of edge (i,j). It requires O(n2) space and is efficient for dense graphs, as checking foran edge is an O(1) operation.
2) Adjacency List: An array of lists, where each index i corresponds to a vertex and stores a list of its neighbors.It requires $O(n+m)$space and is more efficient for sparse graphs.

The practical performance of algorithms like Dijkstra's, whose complexity is often given as $O(m \log n)$, is highly dependent on this choice. For a dense graph, this becomes $O(n^2 \log n)$,whichmaybelessefficientthanasimpler $O(n^2)$implementation.

Therefore, our analysis will consider performance on both sparse and dense graphs, acknowledging the critical interplay between algorithm, data structure, and graph topology.

IV. IN-DEPTH ANALYSIS OF SHORTEST-PATH ALGORITHMS

This section provides a detailed technical examination of each of the four algorithms, covering their operational prin- ciples, formal pseudocode, and a rigorous analysis of their complexity and limitations.

A. *Dijkstra'sAlgorithm*

1) *Principle of Operation:* Dijkstra's algorithm solves the SSSPproblemforaweighteddirectedgraphwithnon-negative edge weights [3], [4]. It operates on a greedy principle, iteratively building a set of vertices, S, for which the shortest path fromthe source $s$is known. Initially, $S$is empty, and the distance to all vertices is set to infinity, except for the source, whose distance is 0 [5].

The algorithm maintains a priority queue,Q, of verticesthat have been discovered but are not yet in S. The priority of each vertex $u$in $Q$is its current known shortest distance from the source, $d[u]$. In each iteration, the algorithm extracts the vertex $u$with the minimum distance from Q, adds it to S, and performs a "relaxation" step for each of its neighbors, v[3], [4]. Relaxationofan edge (u,v)consists ofchecking whether thepathto$v$through$u$isshorterthanthecurrentlyknownpath to v.If$d[u]+w(u,v)<d[v]$, thedistance $d[v]$isupdated,and thepredecessorof$v$issetto$u$[5].Thisprocesscontinuesuntil thepriorityqueueisempty,atwhichpointtheshortestpathsto allreachablevertices have been determined. Thegreedychoice is justified because, with non-negative weights, the first timeapathtoa vertexis finalized(bymovingitfrom$Q$to$S$),it is guaranteed to be the shortest one [3].

2) *Pseudocode:* Algorithm 1 presents the formal pseu- docode for Dijkstra's algorithm using a min-priority queue.

Algorithm 1 Dijkstra's Algorithm DijkstraG,w,sGraph$G=(V,E)$,weightfunction$w$, source vertex sDistances$d[v]$andpredecessors $\pi[v]$forall$v\in V$Initialize-Single-Source$(G,s)$$S\leftarrow\emptyset$$Q\leftarrow V$ Min-priority queueof vertices $Q\neq\emptyset$$u\leftarrow$Extract-Min$(Q)$$S\leftarrow S\cup\{u\}$eachvertex$v\in$Adj$[u]$Relax$(u,v,w)$ Initialize-Single-SourceG,seachvertex$v\in V$$d[v]\leftarrow\infty$
$\pi[v]\leftarrow$NIL$d[s]\leftarrow0$Relax$u,v,w$$d[v]>d[u]+w(u,v)$
$d[v]\leftarrow d[u]+w(u,v)$$\pi[v]\leftarrow u$Decrease-Key$(Q,v,d[v])$

3) *Complexity Analysis:* The time complexity of Dijkstra's algorithm is critically dependent on the implementation of the min-priority queue Q [4].

- Simple Array: If $Q$is an unsorted array, 'Extract-Min' takes $O(n)$ time and 'Decrease-Key' takes $O(1)$ time. Since 'Extract-Min'iscalled $n$timesand 'Relax'(which calls'Decrease-Key') is called at most $m$times, the total complexity is $O(n^2+m)$ $=O(n^2)$. This is efficient for dense graphs where $m \approx n^2$.
- Binary Heap:With a binary heap, both 'Extract-Min'and'Decrease-Key'take$O(\log n)$time.Thetotal complexity becomes$O(n\log n+m\log n)=O((n+ m)\log n)$. For connected graphs ($m\geq n-1$), this is commonly simplified to $O(m\log n)$ [3], [4]. This is the standardimplementationandishighlyefficientforsparse graphs.
- Fibonacci Heap: A more advanced data structure, the Fibonacci heap, provides an amortized time complexityof $O(\log n)$ for 'Extract-Min' and $O(1)$ for 'Decrease- Key'. This yields the best-known theoretical worst-case complexityfor Dijkstra's algorithm: $O(m+n\log n)$ [3], [4].

The space complexity is $O(n+m)$to store the graph (usingan adjacency list) and thedistance/predecessor arrays.

4) *Limitations:* The primary limitation of Dijkstra's algo- rithm is its inability to handle negative edge weights. The greedystrategyreliesontheassumptionthatonceavertex $u$is extracted fromthe priorityqueue, its shortest path $d[u]$is finalized.Anegativeedgecouldviolatethis:apaththrough a later-explored vertex could lead back to $u$or one of its neighborsviaanegativeedge,creatingashorterpaththan the one already found. This invalidates the core assumption of the algorithm [3], [11].

B. *The Bellman-Ford Algorithm*

1) *Principle of Operation:* The Bellman-Ford algorithm solves the SSSP problemin graphs that may contain negative- weight edges [4], [5]. It employs a dynamic programming approach based on the principle of relaxation. The algorithm iteratively relaxes all edges in thegraph.After thefirst pass of relaxing all $m$edges, it guarantees to find all shortest paths oflength1(i.e.,oneedge).Afterthesecondpass,itfinds allshortestpathsoflengthatmost2,andsoon[3].

Since any simple shortest path can have at most $n-1$ edges, the algorithmrepeatsthisrelaxationprocess$n-1$times.After $n-1$iterations, it is guaranteed to have found the shortest path for all reachable vertices, provided there are no negative- weight cycles [3], [4].

AkeyfeatureofBellman-Fordisitsabilitytodetect loop that runs $n-1$ times and an inner loop that iterates throughall $m$edges. This results ina time complexityof $O((n-1)\cdot m)=O(nm)$. The second part, for negative cycle detection, iteratesthroughall $m$edgesonce, taking $O(m)$time.Thus,thetotaltimecomplexityis$O(nm)$ [3], [4]. This holds for both sparse and dense graphs, becoming $O(n^3)$ in the dense case.

Space Complexity: The algorithm requires storage for the distance and predecessor arrays, leading to a space complexity of $O(n)$[4]. If an adjacency list is used forthe graph, the total space is $O(n+m)$.

*C. TheA*SearchAlgorithm*

1) *Principle of Operation:* A* is an informed search algo- rithmthataimstofindtheshortestpathfromasinglesource to a single destination [6], [12]. It can be seen as an extension of Dijkstra's algorithm. Like Dijkstra's, it uses a priority queue (often called the "open set") to explore the graph. However, the priority of a vertex $n$is not just its distance from the source, $g(n)$, but an evaluation function $f(n)=g(n)+h(n)$ [4], [6].

Here, $g(n)$is the cost of the currently known shortest path from the source to n, and $h(n)$is a heuristic function that estimates the cost of the shortest path from $n$to the destina- tion. The heuristic guides the search towards the destination, allowing A* to avoid exploring paths that are moving away fromthegoal[6],[12].Thealgorithmalso maintainsa"closed set" of vertices that have already been fully explored.

For A* to be optimal (i.e., guaranteed to find the shortest path),theheuristicfunction$h(n)$mustbeadmissible,meaning it never overestimates the true cost to reach the goal. That is, $h(n)\leq\delta(n,\text{goal})$ for all vertices n [6], [12]. A common ex- ample of an admissible heuristic is the straight-line Euclidean distance in a geometric pathfinding problem.

2) *Pseudocode:* Algorithm 3 details the A* search proce- dure.

negative-weightcycles.If, after $n-1$iterations, afurther(i.e., n-th) iteration still results in a relaxation (a shortening of a path),itimpliesthatanegative-weightcycleexistsinthegraph and is reachable from the source [3], [5].

3) *Pseudocode:* Algorithm 2 describes the Bellman-Ford procedure, including the step for negative cycle detection.

Algorithm 2 The Bellman-Ford Algorithm Bellman-FordG,w,sGraph$G=(V,E)$,weightfunction w, source vertex s 'true' if no negative cycle, 'false' otherwise. Distances $d[v]$ and predecessors $\pi[v]$. Initialize- Single-Source(G,s) $i\leftarrow 1$ to $|V|-1$ eachedge $(u,v)\in E$ Relax(u,v,w) Standard relaxation each edge $(u,v)\in E$ Check for negative cycles $d[v]>d[u]+w(u,v)$ 'false' Negative cycle detected 'true'

4) *Complexity Analysis:* The structure of the Bellman-Ford algorithm is straightforward, leading to a simple complexity analysis.

Time Complexity: The algorithm consists of two main parts. The first part has a nested loop structure: an outer

Algorithm 3 The A* Search Algorithm A-Star$start,goal,h$Startvertex$start$,goalvertex $goal$,heuristic$h$Theshortestpathfrom$start$to$goal$openSet $\leftarrow\{start\}$Priority queue with f-scores cameFrom$\leftarrow$anemptymap$gScore[v]\leftarrow\infty$for all$v\in V$$gScore[start]\leftarrow 0$$fScore[v]\leftarrow\infty$for all$v\in V$$fScore[start]\leftarrow h(start)$openSetisnot empty$current\leftarrow$vertexinopenSetwiththelowest $fScore$$current=goal$reconstructpath(cameFrom, current) Remove $current$from openSet each neighbor of$current$$tentativegScore\leftarrow gScore[current]+w(current,neighbor)$ $tentativegScore<gScore[neighbor]$cameFrom[$neighbor$]$\leftarrow current$ $gScore[neighbor]\leftarrow tentativegScore$ $fScore[neighbor]\leftarrow gScore[neighbor]+h(neighbor)$ $neighbor$not in openSet Add $neighbor$to openSet failure No path found

5) *ComplexityAnalysis:*TheperformanceofA*ishighly sensitive to the quality of its heuristic.

- Time Complexity: In the worst case, with a non- informative heuristic (e.g., $h(n)=0$for all n), A* becomes equivalent to Dijkstra's algorithm, and its time complexityis$O(m\log n)$with abinaryheap[6].As the heuristic becomes more accurate, the number of explored nodes decreases, and the performance improves significantly.Withaperfectheuristic($h(n)=\delta(n,\text{goal})$), A*onlyexploresnodesontheoptimalpath,leadingto a complexity close to $O(n+m)$in practice.

- Space Complexity: A* must storeall generated nodes in theopenandclosedsets.Intheworstcase,thiscanbethe entire graph, leading to a space complexity of $O(n+m)$. For problems with large state spaces, such as in game AI or robotics, this exponential space requirement can be a significant drawback [5], [6].

D. TheFloyd-WarshallAlgorithm

1) *Principle of Operation:* The Floyd-Warshall algorithm solves the APSP problem using a dynamic programming approach [4], [13]. It works by iteratively considering each vertex in the graph as a potential intermediate vertex in the shortest paths between all other pairs of vertices.

Let$d^{(k)}_{ij}$bethe weightoftheshortestpathfromvertex $i$tovertex$j$usingonlyintermediateverticesfromtheset $\{1,2,...,k\}$.Thealgorithmcomputesasequenceof$n\times n$matrices,$D^{(0)}$,$D^{(1)}$,...,$D^{(n)}$,where$D^{k}[i,j]=d^{k}$.The

2) *Negative Cycle Detection:* The Floyd-Warshall algo- rithm can be used to detect negative-weight cycles. After the algorithmcompletes, ifanydiagonal element *dii*is negative,it indicatesthatvertex$i$ispartoforcanreachanegative-weight cycle [4], [11]. This is because the path from a vertex to itself should be 0, and a negative value implies that the path can be made shorter by traversing a cycle.

V. COMPARATIVE ANALYSIS AND APPLICATIONS

The choice of a shortest-path algorithm is a critical design decision that hinges on the specific constraints of the prob- lem. This section provides a direct comparison of the four algorithms based on the framework established earlier and discusses their principal real-world applications. A summary of this comparison is presented in Table I.

A. SSSP vs. APSP

The most fundamental distinction is between algorithmsthat solve the SSSP problem and those that solve the APSP problem. For APSP, one could simply runan SSSP algorithm, like Dijkstra's, once for each of the n vertices.

1) Forsparsegraphs($m\approx n$)**:**RunningDijkstra's$n$times yieldsatotaltimecomplexityof$n\times O(m\log n)$=

ij

(0)

$O(nm\log n)$.ThisisgenerallymoreefficientthanFloyd-

basecase,D ,istheinitialadjacencymatrixofthegraph.

Warshall's$\Theta(n^3)$complexity.Ifnegativeweightsare

Thecoreofthealgorithm isthefollowingrecurrencerelation present,Johnson'salgorithm,whichusesBellman-Ford

[4]:

(k)

($k-1$)

($k-1$)

($k-1$)

onceandDijkstra$n$times,achieves$O(nm+n^2\log n)$

$$d^{(k)}_{ij}=\min(d^{(k-1)}_{ij},d^{(k-1)}_{ik}+d^{(k-1)}_{kj})$$

This formula states that the shortest path from $i$to $j$using intermediate vertices up to $k$is either the shortest path using onlyvertices upto $k-1$, or it is the path that goes from$i$to$k$andthenfrom$k$to j(bothusingintermediateverticesup to $k-1$). By iterating $k$from 1 to n, the final matrix $D^{(n)}$containstheshortest-path weightsbetweenallpairsofvertices [13].

2) *Pseudocode:*TheimplementationoftheFloyd-Warshall algorithm is notably compact, as shown in Algorithm 4.and is also superior to Floyd-Warshall on sparse graphs [3].

- For dense graphs **(**$m\approx n^2$**):** RunningDijkstra'sn times results in $O(n\cdot n^2)=O(n^3)$ with a simple array priority queue, or$O(n\cdot n^2\log n)$withabinaryheap.Inthis sce- nario, Floyd-Warshall's $\Theta(n^3)$complexity is competitive andoftenpreferredduetoitssimplerimplementationand better memory locality [4], [13].

B. Handlingof NegativeWeights

Algorithm4TheFloyd-WarshallAlgorithm

Floyd-Warshall$W$ $n\times n$weightmatrix$W$ofagraphwith$n$ vertices$n\times n$matrixofshortest-pathweights$D^{(0)}$$W$$k\leftarrow 1$ to$n$ $i\leftarrow 1$ to$n$ $j\leftarrow 1$ to$n$ $d^{k)}\leftarrow \min(d^{(k-1)},d^{(k-1)}+$

Thepresenceofnegativeedgeweightsisamajordifferen- tiating factor.

- Dijkstra'salgorithmfails becauseitsgreedyassumption

$d^{(k-1)})D^{(n)}$ ij ij ik isviolated.Itisfundamentallyunsuitedforthisproblem

kj

3) ComplexityAnalysis:

- Time Complexity: The algorithm's structure consists of three nested loops, each iterating from 1 to n. The inner operationisconstanttime.Therefore,thetimecomplexity is consistently $\Theta(n^3)$regardless of the graph's structureor density [4], [13].
- Space Complexity: The algorithm requires an $n\times n$ matrix to store the distances. With a space-optimized implementationthatupdatesthematrixinplace,thespace complexity is $\Theta(n^2)$[4]. An additional $\Theta(n^2)$matrix can be used to reconstruct the paths themselves. domain.
- Bellman-Ford is the classic SSSP solution for graphs withnegativeweights.Itsiterativenatureensuresthat it correctly propagates the effects of negative edges throughoutthegraph[4],[5].Itsabilitytodetectnegative cycles is crucial in applications where such cycles render the problem ill-defined, such as in economic arbitrage analysis [3].
- Floyd-Warshall also correctly handles negative weights for the APSP problem and provides a simple mechanism for detecting the presence of any negative cycle in the graph by inspecting the diagonal of the final distance matrix [4].

TABLEI
COMPARATIVESUMMARYOFFOUNDATIONALSHORTEST-PATHALGORITHMS

Criterion	Dijkstra'sAlgorithm	Bellman-FordAlgorithm	A*SearchAlgorithm	Floyd-WarshallAlgorithm
ProblemType	SSSP	SSSP	Single-Pair(typically)	APSP
AlgorithmicParadigm	Greedy	DynamicProgramming	Heuristic Search (Informed)	DynamicProgramming
EdgeWeights	Non-negativeonly	Positiveornegative	Non-negative(typically)	Positiveornegative
NegativeCycle	Cannothandle	Detectsreachable cycles	Notdesignedforthis	Detectsallcycles
TimeComplexity	$O(m\log n)$(BinaryHeap)	$O(nm)$	Varies(heuristic-dependent)	$\Theta(n^3)$
SpaceComplexity	$O(n+m)$	$O(n+m)$	$O(n+m)$(canbeexponential)	$\Theta(n^2)$
PrimaryApplication	Networkrouting(OSPF),GPS	Distance-vectorprotocols(RIP)	GameAI,Robotics,Pathfinding	All-pairsanalysis,Transitiveclosure

C. Heuristicvs.UninformedSearch

A* stands apart from Dijkstra's by incorporating domain knowledge.WhileDijkstra'salgorithmexploresoutwardsuni- formly from the source, A* focuses its search towards thegoal[6],[12].Thismakesitexceptionallypowerfulforsingle- pair pathfinding in large state spaces, such as maps or game levels.Thetrade-offisthecostofdesigningandcomputing an effective heuristic. A poor heuristic can make A* perform no better, or even worse, than Dijkstra's, while a highly accurate heuristic can lead to near-optimal search performance.

D. Real-WorldApplications

The theoretical differences between the algorithms directly map to their suitability for different real-world applications.

1) Dijkstra'sAlgorithm:Itsefficiencyandsimplicitymake it the standard for SSSP problems with non-negative costs. It is famously used in the OSPF (Open Shortest Path First) internet routing protocol, where link costsare positive metrics [3]. It is also fundamental to GPS navigation and mapping services for calculating the fastest or shortest route between two points [4], [5].

2) Bellman-Ford Algorithm: Its ability to handle negative weights makes it suitable for distance-vector routing protocols like the Routing Information Protocol (RIP). While RIP itself does not use negative weights, the Bellman-Ford structure is robust to the dynamic updates and potential routing loops that can occur in such net- works [3], [4]. It is also used in analyzing networks for arbitrage opportunities where negative cycles represent profitable loops.
3) A*SearchAlgorithm:A*isthedominantalgorithm in any domain requiring goal-directed pathfinding. This includes video game development for NPC (non-player character) pathfinding, robotics for navigation and mo- tion planning, and logistics applications for finding op- timal routes for autonomous vehicles like drones or warehouse robots [4], [6].
4) Floyd-WarshallAlgorithm:Its$\Theta(n^3)$complexitylimits

its use to smaller or denser graphs. It is valuable for computingthe**transitive closure** ofagraph(determining reachability between all pairs of nodes) [11]. Other ap- plications include computing all-pairs distances in social networks to find the ”degrees of separation” between all members, or in bioinformatics for sequence alignment [3], [4].

VI. CONCLUSION AND FUTURE DIRECTIONS

This review has systematically analyzed four of the most foundational algorithms for solving the shortest-path prob- lem: Dijkstra’s, Bellman-Ford, A*, and Floyd-Warshall. The analysis demonstrates that no single algorithm is universally superior.Instead,eachrepresentsauniquesolutiontailored to a specific set of problem constraints, embodying a distinct trade-off between speed, generality, and problem scope.

Dijkstra’s algorithm offers unparalleled efficiency for the SSSP problem on graphs with non-negative weights, makingit the default choice for a wide range of applications. The Bellman-Ford algorithm, while slower,providesthe necessary robustness to handle negative edge weights and the critical capability to detect negative-weight cycles. The A* search algorithm showcases the power of heuristic guidance, dra- matically accelerating single-pair pathfinding in applications wheredomainknowledgecanbeleveragedtodirectthesearch. Finally, the Floyd-Warshall algorithm provides a comprehen- sive, albeit computationally intensive, solution for the APSP problem, proving invaluable for dense graphs and problems requiring a complete distance matrix.

The true legacy of these algorithms extends beyond their directapplications.Theyserveasfundamental buildingblocks andconceptualbaselinesfortheongoingdevelopmentofmore advancedpathfindingtechniques.Thisisevidentinseveralkey areas of modern research:

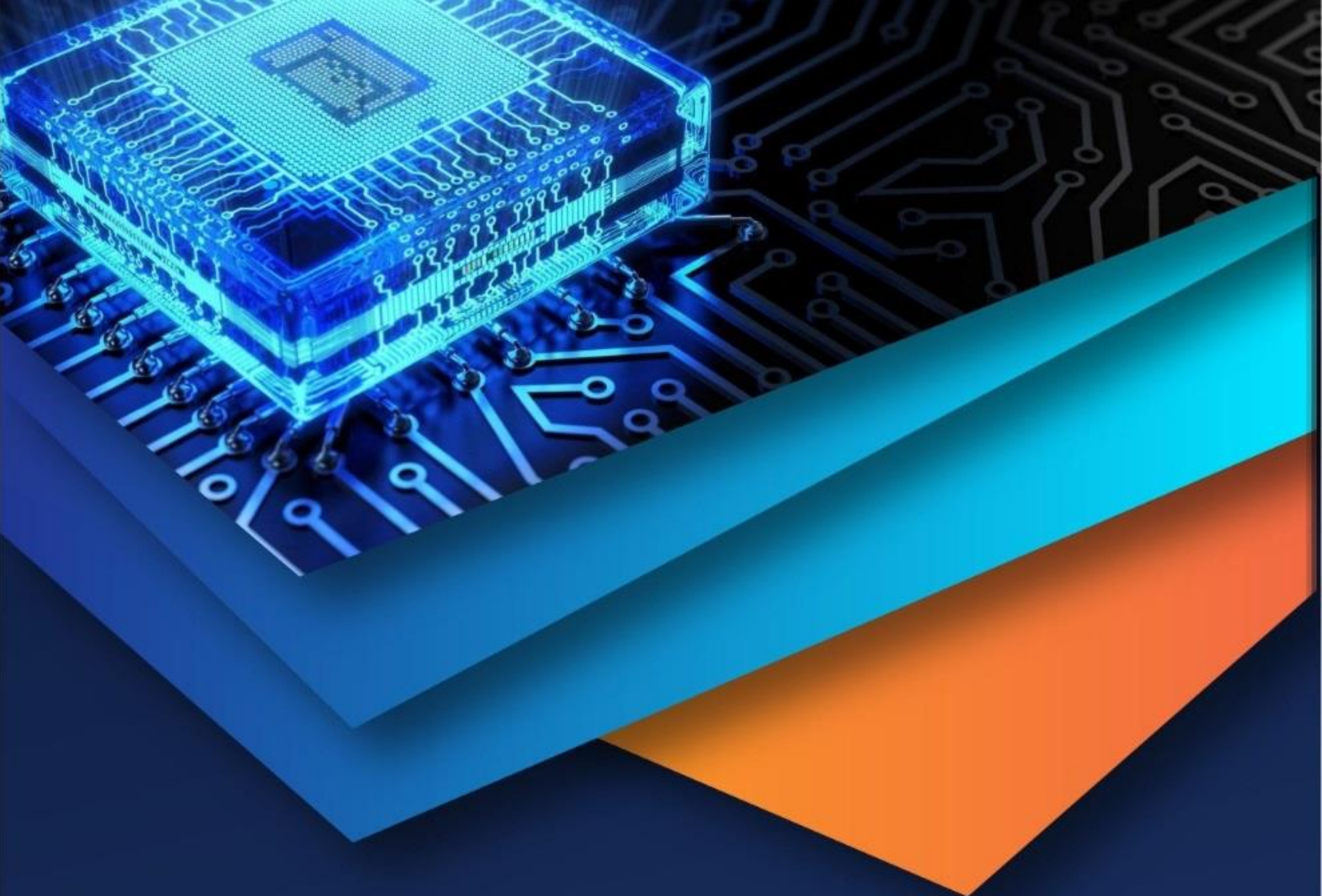
- HybridAlgorithms:Moresophisticatedalgorithmsoften combine the strengths of these foundational methods. A prime example is Johnson’s algorithm, which cleverly uses Bellman-Ford as a preprocessing step to re-weight a graph, eliminating negative edges. This allows the more efficient Dijkstra’s algorithm to be run subsequently for each vertex, making it one of the fastest solutions for APSP on sparse graphs with negative weights [3].
- Parallelization: As datasets grow, parallel computation becomes essential. The inherent structure of these algo- rithms dictates their suitability for parallelization. The nested loops of Floyd-Warshall, for instance, are highly amenable to parallel execution through techniques like 2D block mapping [1], [2]. In contrast, the inherently sequential nature of Dijkstra’s algorithm—where each step depends on the minimum-priority vertex from the previous step—presents a greater challenge for parallel implementation [2], [5].
- New TheoreticalFrontiers:Evenafterdecadesofstudy, thetheoreticallimitsofshortest-pathcomputationarestill being explored. Recent work by Duan et al. has intro- duced a novel algorithm that breaks the long-standing ”sorting barrier” associated with Dijkstra’s algorithm, achieving a slightly faster time complexity for the SSSP problem on directed graphs [5], [8]. This breakthrough, which itself combines ideas from both Dijkstra’s and Bellman-Ford’s algorithms, demonstrates that this field remains a vibrant area of active research.

In conclusion, the algorithms of Dijkstra, Bellman, Ford, Hart, Nilsson, Raphael, Floyd, and Warshall are not merely historical artifacts. They are enduring pillars of computer sciencethatcontinuetosolvecountlessreal-worldprob- lems today. More importantly, they provide the core princi- ples—greedyselection,iterativerelaxation,dynamicprogram- ming,and heuristicguidance—that inspireandenablethe next generation of algorithms designed to navigate the ever-more- complex networks of our digital world.

REFERENCES

[1] A.Madkour,W.G.Aref,F.U.Rehman,M.A.Rahman,and S. Basalamah, “A survey of shortest-path algorithms,” arXiv preprintarXiv:1705.02044,2017.
[2] L.dosSantosandL.M.deAssis,“Parallelizationofshortestpathclassalgorithms:acomparativeanalysis,”PesquisaOperacional,vol.43,2023.
[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introductionto Algorithms, 3rd ed.MIT Press, 2009.

[4] R. Sedgewick and K. Wayne, Algorithms, 4th ed. Addison-WesleyProfessional, 2011.
[5] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest pathsalgorithms: Theory and experimental evaluation," Mathematical pro-gramming, vol. 73, no. 2, pp. 129–174, 1996.
[6] N. J. Nilsson, Principles of artificial intelligence. Tioga PublishingCompany, 1980.
[7] E.W.Dijkstra,"Anoteontwoproblemsinconnexionwithgraphs," NumerischeMathematik,vol.1,no.1, pp.269–271,1959.
[8] R. Duan, G. He, M. Lyu, Y. Wu, and R. Xie, "Breaking the sortingbarrier for shortest paths on directed graphs," in Proceedings of the2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).SIAM, 2023, pp. 136–152.
[9] R.Bellman, "Onaroutingproblem,"Quarterlyof AppliedMathematics,vol. 16, no. 1, pp. 87–90, 1958.
[10] L. R. Ford, Jr., "Network flow theory," RAND Corporation, Tech. Rep.P-923,1956.
[11] S. Hougardy, "The floyd-warshall algorithm on graphs with negativecycles," Information Processing Letters, vol. 110, no. 8-9,pp. 279–281,2010.
[12] P.E.Hart,N.J.Nilsson,andB.Raphael,"Aformalbasisfortheheuristicdetermination of minimum cost paths," IEEE Transactions on SystemsScience and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.
[13] R. W. Floyd, "Algorithm 97: Shortest path," Communications of theACM, vol. 5, no. 6, p. 345, 1962.
[14] B.Roy,"Transitivite´etconnexite´,"ComptesRendusdel'Acade´miedesSciences, vol. 249, pp. 216–218, 1959.
[15] S. Warshall, "A theorem on boolean matrices," Journal of the ACM,vol. 9, no. 1, pp. 11–12, 1962.



10.22214/IJRASET

INDEX COPERNICUS
45.98

IMPACT FACTOR:
7.129

TOGETHER WE REACH THE GOAL
IMPACT FACTOR:
7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY