International Journal on
"Technical and Physical Problems of Engineering"
(IJTPE)
Published by International Organization of IOTPE

ISSN 2077-3528
IJTPE Journal
www.iotpe.com
ijtpe@iotpe.com

June 2022 | Issue 51 | Volume 14 | Number 2 | Pages 240-247

SIMULATION AND COMPARISON OF VARIOUS SHORTEST PATH ALGORITHMS WITH MOBILE ANDROID APPLICATION

U. Hacizade [1] G. Bal [2]

1. Computer Engineering Department, Halic University, Istanbul, Turkey, ulviyehacizade@halic.edu.tr
2. Vektora Information Technologies, Istanbul, Turkey, giraybal@gmail.com

Abstract- In this study, the properties and differences of some commonly used shortest pathfinding algorithms were examined. Thus, the performances of these algorithms were compared in different situations on the maps required by the game and the use of the most appropriate algorithms for these situations was suggested. These comparisons were performed on an Android-based mobile application in order to show them to a wide range of users. By taking the results of this study into account and choosing the correct shortest path algorithm, the developer can minimize the memory requirements on the system where the algorithm will be used. Also, use of a suitable algorithm provides a significant time advantage.

Keywords: Shortest Pathfinding, Android, Dijkstra's Algorithm, Bellman-Ford Algorithm, Floyd-Warshall Algorithm, A* Algorithm.

1. INTRODUCTION

Nowadays, when almost all people in the world have at least one technological device, technology is advancing very rapidly. As a result, the use of portable devices such as small-sized mobile phones, tablets, wearable technology such as smartwatches has increased incrementally compared to the past [1]. With these technologies developed by free software groups, Android-based phones used by billions of people today also show a rapid increase in usage. Nowadays researches are carried out to ensure effective and fast execution of especially strategy games and navigation-based applications on phones.

The shortest path problem is one of the most practical problems in network analysis. Based on the various mathematical models, different algorithms have been proposed for optimal routing, given the network's parameters, characteristics, and structure. There are some commonly used shortest pathfinding algorithms in literature: Dijkstra algorithm [2, 3, 4, 5, 6]; Bellman-Ford algorithm [7, 8, 9]; Floyd-Warshall algorithm [10, 11, 12] and A* algorithm [13, 14]. These algorithms were used in the developed Android-based mobile applications [15, 16, 17].

The problem of determining the shortest distance between the participants of the transportation process can be achieved using graph theory [18]. If number of variables in the optimization problem is continuous and number of them is discontinued, then Genetic Algorithms (GA) can be used [19].

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. This algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956 and published in 1959 [2]. The original variant of Dijkstra's algorithm found the shortest path between two nodes. A more common variant fixes a single node as the "source" node and finds the shortest paths from the source to all other nodes in the graph that produces a shortest-path tree.

It is shown in [3] that Dijkstra's algorithm is strongly inspired by Bellman's Principle of Optimality and constitutes a dynamic programming successive approximation procedure par excellence both conceptually and technically. Consequently, this algorithm can be incorporated into dynamic programming. In [4], the Dijkstra algorithm is used in robot path planning. The shortest path is selected in the process of the barrier. An idea of shortlisting the appropriate nodes in a graph is proposed in [5], which is then used to find the shortest path via Dijkstra's algorithm. In [6], the Dijkstra algorithm was selected for the shortest path data analysis and route selection.

Bellman-Ford's algorithm can solve the single-source shortest path (SSSP) problem and better applies parallelization for multi-core architectures. The study [7] presents a parallel implementation of the Bellman-Ford algorithm that exploits the architectural characteristics of recent graphics processing unit (GPU) architectures to improve both performance and work efficiency. This work presents different optimizations to the implementation, which sufficiently reduces the redundant work caused by the parallelization.

The Bellman-Ford algorithm has been applied in fuzzy networks, for the past few years. In [8], the neutrosophic version of Bellman's algorithm based on the trapezoidal neutrosophic numbers is proposed. In [9], the routing method based on Bellman-Ford algorithm is proposed within the Software-Defined Networking (SDN). The SDN data were represented in the graph with nodes and vertices.

The Floyd-Warshall algorithm is founded by Robert W. Floyd in 1967. It is a simple algorithm to compute the shortest paths between all pairs of vertices in an edge-weighted directed graph. The number of side weights on a path is the weight of the path. This algorithm calculates the least weights of all paths connecting a pair of points and does so simultaneously for all pairs of points. The Floyd-Warshall algorithm compares all possible paths in the graph for each side of all nodes. This may be due to the evaluation of decision making (choosing the shortest path) at each stage between two vertices until the estimate becomes the optimal value [10]. It can also be used to detect the negative cycles. It is shown in [11] that, for this problem, many existing implementations of the Floyd-Warshall algorithm will fail because exponentially large numbers can appear during its execution.

In [12], the problem of finding the route is presented as a multi-vehicle route problem with a time window. The dynamic programming-based algorithms to overcome the problem of conflict-free routes using time windows are described in this work. A* is a graph traversal and path search algorithm. This algorithm is used in many areas of computer science due to its completeness, optimality, and optimal efficiency [13]. The A* is a best-first search algorithm, which means that it is formulated in terms of weighted graphs: starting from a certain starting node of the graph, it seeks to find a path to a given target node at the lowest cost.

An A* search algorithm was applied to path planning in a Chinese chess game in [14]. Based on the results of this work, the shortest path was found for mobile robots (chess pieces) moving to target points from their starting points in a collision-free environment. The paper [20] presents a simplified path solving algorithm for known and unknown environments. In the study robot range is limited to the size of the map created within the computer's internal memory. Two path planning algorithms (Dijkstra algorithm and A* algorithm) have been simulated to analyze the most efficient execution. The obtained results show that the A* algorithm gives better performance and lower computational cost in comparison with the Dijkstra algorithm.

Different types of pathfinding problems can be solved by various algorithms, e.g., single-source shortest path problems by Dijkstra's algorithm, single-source problems if the edge weights are negative by Bellman-Ford algorithm, single-pair shortest path problems by A* search algorithm using heuristics to try to speed up the search, and all-pairs shortest paths problems by Floyd-Warshall algorithm.

In this study, the properties and differences of some of the most commonly used shortest pathfinding algorithms such as the Dijkstra algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm, and A* algorithm were examined, and these algorithms were used in the developed Android-based mobile application and their performances were compared. Thus, the performances of these algorithms were compared in different situations on the maps required by the game and the use of the most appropriate algorithms for these situations was suggested.

These comparisons were performed on an Android-based mobile application in order to show them to a wide range of users.

2. INVESTIGATION OF VIDEO GAMES AND NAVIGATION PROGRAMS USING SHORTEST PATH ALGORITHMS

Video games are one of the entertainment activities used especially by children and young people since the entrance of computers into homes. The shortest pathfinding algorithms are the algorithms in which the grid logic and game maps are divided into certain squares especially in strategy games and weighted or unweighted graphs are used for the purpose of the game. In these games, the Dijkstra and other algorithms, especially the A* algorithm are used. Some examples of strategy games are Dune 2000 [21], a video strategy game published by Westwood in 1998; Age of Empires [22], also a video strategy game developed by Microsoft Game Studios in 1998.

Another area where shortest path algorithms are used is navigation applications. In these applications, the coordinates of the users are shown on the map. When users select a destination, the application draws a path from the user's point to the destination point. This road is located at the intersection of roads with given costs. The cost of graphs is indispensable for navigation applications. Especially on roads with traffic, this cost value will be more. Users do not see any point and cost values on the map when they look at the application, but the shortest pathfinding algorithms are used to go from one point to another point in the background [1].

2.1. Dijkstra Algorithm

Dijkstra algorithm is an algorithm that finds the shortest distance from one point to another point on roads with certain metric weighted values [2]. It was developed in 1956 by Dutch computer scientist and mathematician Edsger W. Dijkstra.

The working principle of this algorithm is; determine a "source" point and find the shortest path by passing over the costs of all other points from this source point (Figure 1). As shown in Figure 1, the Dijkstra algorithm is an algorithm that can find the distances from one point to many points at one time. Some conditions must be met in order to use the algorithm in a healthy way. If one of the costs is given a negative value, the results in the algorithm will be inaccurate and inconsistent, and the resulting paths will not be the shortest. If it is desired to give negative values to the costs on the roads, the "Bellman-Ford" algorithm can be used.

Dijkstra algorithm is a greedy algorithm. The pseudocode of the algorithm is shown in Figure 2. The mobile Android application has been developed by adhering to this pseudocode.

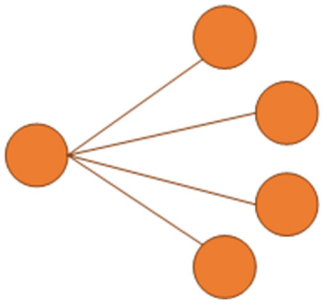

Figure 1. Graph of going from one point to many points

```
function dijkstra(G,S):

    create vertex set Q

    for each vertex v in G:          // Loop for all points on the graph
        distance[v] <- INFINITY      // Write infinite because the distance from the
                                     // source point to the destination point is unknown
        previous[v] <- UNDEFINED     // Write infinite to the best shortest path found in the previous point
        add v to Q                   // Make all points unvisited

    distance[S] <- 0                         // Distance from source to source is 0

    while Q is not empty:                    // Run loop until unfinished points are over
        u <- vertex in Q with min distance[u] // Find and select the shortest point

        remove u from Q                      // Delete this point from the list of unvisited points

        for each neighbor v of u:            // loop for all neighbor points of 'u'
            alt <- distance[u] + length(u, v)
            if alt < distance[v]:            // Is this shortest path to point 'v'?
                distance[v] <- alt           // Update distance value
                previous[v] <- u             // Set the previous point

    return distance[], previous[]            //return points and previous point list
```

Figure 2. Pseudocode for Dijkstra algorithm

2.2. Floyd-Warshall Algorithm

Floyd-Warshall algorithm is another algorithm for finding the shortest path on a graph. The algorithm was published in 1962 by Robert Floyd. The algorithm, which was published several years before Bernard Roy in 1959 and Stephen Warshall in 1962, is now known as the Floyd-Warshall algorithm. The algorithm is also named as Floyd algorithm, Roy-Warshall algorithm, Roy-Floyd algorithm, and WFI algorithm [23].

The difference between this algorithm and the Dijkstra algorithm is that the paths can have positive or negative values. However, the path to be used in the algorithm from the same point to the same point, for example, from point A to point A loop path should not take a negative value. A path from one point to another may take a negative value too. As shown in Figure 3, the Floyd-Warshall algorithm is an algorithm that can find distances from many points to many points at once.

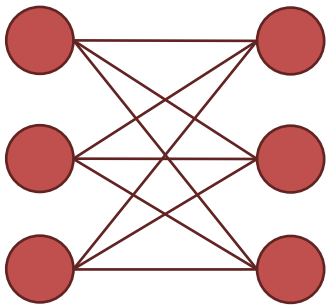

Figure 3. Graphic of going from many points to many points

The pseudocode of the algorithm is shown in Figure 4. The mobile Android application has been developed by adhering to this pseudocode.

```
for i = 1 to N // Start loop for all points
    for j = 1 to N // Start second loop to compare all points
        if there is an edge from i to j // If there is a path from point i to point j
            distance[0][i][j] = the length of the edge from i to j // add this path to array
        else
            distance[0][i][j] = INFINITY // Set infinity to between two points

for k = 1 to N // Start loop for all points
    for i = 1 to N // Start second loop to compare all points
        for j = 1 to N // Start thitd loop to compare all points
            distance[k][i][j] = min(distance[k-1][i][j], distance[k-1][i][k] + distance[k-1][k][j])
            // Compare previous calculated distance if distance between points is shorter
            // If the new distance found is shorter then the path will be shorter and select it

return distance
```

Figure 4. Pseudocode for Floyd-Warshall algorithm

2.3. Bellman-Ford Algorithm

The Bellman-Ford Algorithm is a weighted graphical algorithm that finds shortcuts from one source point to all other destination points. This algorithm was first introduced by Alfonso Shimbel in 1955 and later published in 1956 and 1958 by Richard Bellman and Lester Ford, respectively. The algorithm published by Edward F. Moore in 1957 is also referred to as the Bellman-Ford-Moore algorithm in some sources [24]. This algorithm runs slower than the Dijkstra algorithm. However, besides slower operation, there are some advantages, for example, some of the paths given to the algorithm may take negative values. Negative weighted paths are used in many applications, and the algorithm can detect the presence or absence of negative loops on the graphic.

As shown in Figure 5, Bellman-Ford is an algorithm that can find the distances from one point to many points at one time even if it is negative.

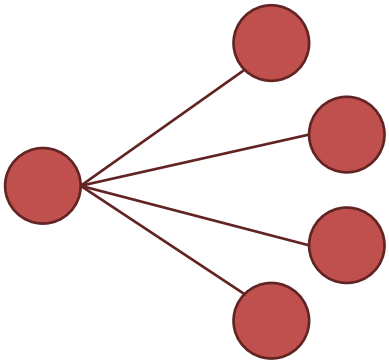

Figure 5. Graphic of going from one point to many points

The pseudocode of the algorithm is shown in Figure 6. The mobile Android application has been developed by adhering to this pseudocode.

```
function bellmanFord(G,S) // G graph, S source point
  for each vertex V in G // Loop for all points on the graph
    dist[V] <- infinite // Set infinite to start distance values
    prev[V] <- NULL // Make the prev point null
  dist[S] <- 0 // Set the table value of the source point to 0
  for each vertex V in G // Start loop for all points on the graph
   for each edge (U,V) in G // Loop for all paths from U to V
    tempDistance <- dist[U] + edge weight(U,V) // Set temporary distance
    if tempDistance < dist[V] // If the newly calculated distance is shorter than the old one
      dist[V] <- tempDistance // Set new distance value to shortest
      prev[V] <- U // Determine from which point this new shortest is coming

  for each edge (U,V) in G                // Loop for all paths from U to V
   if dist[U] + edge weight(U, V) < dist[V] // If the weight sum is less than the distance
      Error: Negative Cycle Exists        // The negative cycle found. Loop ends.

  return dist[], prev[] //return points and prev point list
```

Figure 6. Pseudocode for Bellman-Ford algorithm

2.4. A* Algorithm

Published by Peter Hart, Nils Nilsson, and Bertram Raphael of Stanford Research Institute in 1968, this algorithm is one of the most popular shortest pathfinding algorithms because of its speed. This algorithm, which is much preferred in video games, is called smarter than the other algorithms according to the solution style that results from the problem-solving technique. The algorithm uses the data it holds efficiently at every point. When the algorithm is executed, it adds an "F" value by adding two different parameters, "G" and "H", at each step. At each step, it selects the smallest of the resulting "F" values and proceeds to the next step.

The value G here corresponds to the cost between the currently selected point and the point the algorithm is targeting for the next step. The H value corresponds to the estimated cost value between the currently selected point and the endpoint. As shown in Figure 7, A* Algorithm is an algorithm that finds the distance from one point to another point at once.

Figure 7. Graphic of going from one point to one point

The pseudocode of the algorithm is shown in Figure 8. The mobile Android application has been developed by adhering to this pseudocode.

```
function AStar(start, finish)
   closedList :=            // Set closed list
   openList := (start)      // Set open list
   cameFrom := an empty map // Set cameFrom list
   gScore := map with default value of Infinity // Define the sequence holding G values
   gScore[start] := 0 // Make G value of source point to 0
   fScore := map with default value of Infinity // Define sequence holding F values
   while openList is not empty // Loop as long as there is a point in the open list
      current := // The present point with the least value of F
      if current = finish // Finish the loop if arrived to finish
         return reconstruct_path(cameFrom, current)
      openList.Remove(current) // Remove current point from open list
      closedList.Add(current ) // Add current point to closed list
      for each neighbor of current // Loop for all neighbors of the current point
         if neighbor in closedList continue
         if neighbor not in openList openList.Add(neighbor)
         tentative gScore := gScore[current] + dist between(current, neighbor)// Calculate G
         if tentative gScore >= gScore[neighbor] continue // Continue if value is not smaller
         cameFrom[neighbor] := current //set point as landing point
         gScore[neighbor] := tentative gScore // Save G and F
         fScore(neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, finish)
   return failure
function reconstruct_path(cameFrom, current) // Find the shortest path in array
   total_path := [current]
   while current in cameFrom.Keys: // Loop as long as it have cameFrom point
      current := cameFrom[current] // Update current point
      total_path.append(current ) // Add current point to short path
   return total_path // Return found shortest path sequence
```

Figure 8. Pseudocode for A* Algorithm

3. COMPARISON OF VARIOUS SHORTEST PATH ALGORITHMS VIA MOBILE ANDROID APPLICATION

3.1. Map Selection Screen

In the map selection screen of the developed Android-based mobile application, it is possible to select a common map on which the algorithms will work. The different pre-prepared maps selected here contain start points, endpoints, and obstacles. There are 4 maps embedded in the application. The map selection screen is shown in Figure 9. The letter "S" on this screen means "Single Source". "M" letter means "Multiple Sources". For example, the map indicated by "S->S" means "Single Source to Single Source". Map with "Negative Cycle" indicates negative loops in the content of the map.

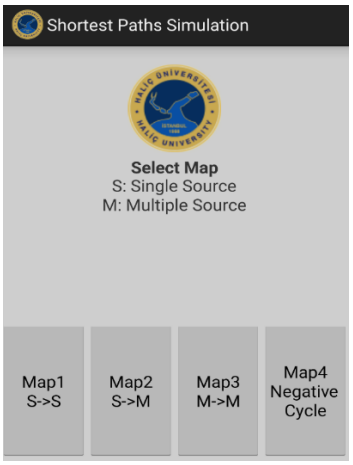

Figure 9. Map selection screen

3.2. Algorithm Selection Screen

After selecting the map, 4 different shortest pathfinding algorithms appear on the application. One of these algorithms can be selected regardless of the order. The algorithm selected from this screen will be used in the simulation screen with the map selected from the map screen. When the "Results" button is pressed, the results screen is displayed after the algorithms are executed sequentially. The algorithm selection screen is shown in Figure 10.

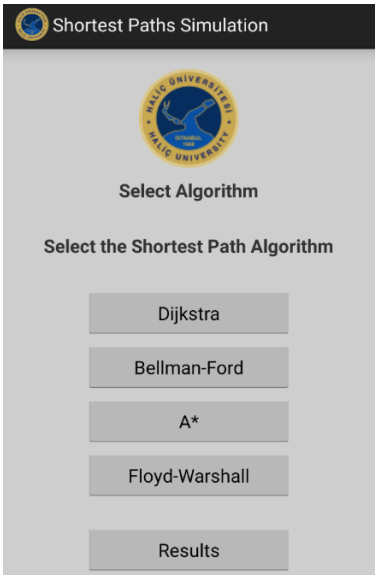

Figure 10. Algorithm selection screen

3.3. Simulation Screen

The selected algorithm is displayed at the top of the simulation screen. In the center of the screen, the map selected from the map screen has a grid of squares. When the selected algorithm is executed in this section, the shortest path that the algorithm finds will be displayed step by step with green squares on the map in the middle of the screen. The algorithm is activated by pressing the "Start" button at the bottom of the screen. Dijkstra algorithm simulation process is shown in Figure 11. The shortest path will work with the start point at the top left and the endpoint at the bottom right.

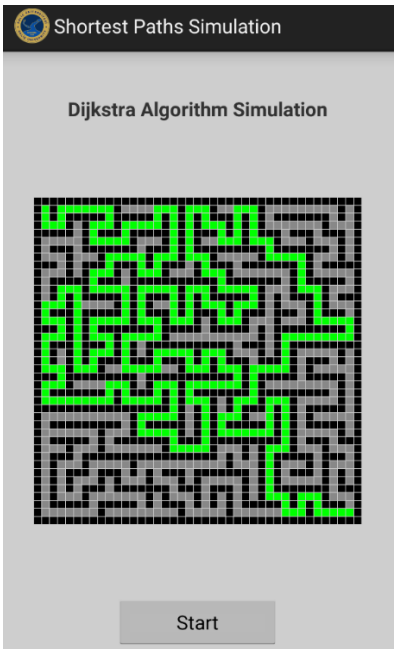

Figure11. Simulation screen

3.4. Results Screen

When "Map 1" is selected from the map screen, each of 4 algorithms mentioned is used in order to calculate the shortest route from one point to another point on this map. Some algorithms run slower or faster than others. These speeds, which vary depending on the design of the algorithm, present the results of this comparison made to the person using the algorithm, and these results are displayed on the results screen.

After selecting "Map 1" on the map selection screen shown in Figure 9, one of the listed algorithms Dijkstra, Bellman-Ford, A* or Floyd-Warshall is selected in the algorithm selection screen shown in Figure 10, and the simulation screen shown in Figure 11 is opened. As shown in Figure 11, pressing the "Start" button starts the execution of the algorithm. When the selected algorithm completes finding the shortest path, this path is shown with green boxes as shown in Figure 11. The results for "Map 1" where the calculations are made from a single source to a single source for each algorithm are shown in Figure 12.

Map 1 Results
(Single Source to Single Source)

	Duration	Length
Dijkstra	154ms	377
Bellman-Ford	10180ms	377
A Star	78ms	377
Floyd-Warshall	61518ms	377

Figure 12. Calculated results of Map 1

As shown in Figure 12, the "Duration" column is the runtime of the algorithm. The unit of time used here is millisecond (ms). The "Length" column shows the length of the shortest path that the algorithm can find.

When "Map 2" is selected in Figure 9, the application starts to find the shortest path from a single source to multiple sources on this map at the same time. When all algorithms are selected from the screen in Figure 10 and run one by one, all results are transferred to the results screen again. The results for "Map 2" where the calculations are made from a single source to multiple sources for each algorithm are shown in Figure 13.

Map 2 Results
(Single Source to Multiple Source)

	Duration	Length
Dijkstra	127ms	-
Bellman-Ford	14144ms	-
A Star	18703ms	-
Floyd-Warshall	61105ms	-

Figure 13. Calculated results of Map 2

When "Map 3" is selected in Figure 9, calculations are made from many points to many points on this map at the same time. The results for "Map3" where the calculations from multiple sources to multiple sources are made for each algorithm are shown in Figure 14.

As seen in Figure 13 and Figure 14, the "Length" columns for "Map 2" and "Map 3" are empty. Since the calculations made on these maps are obtained from one point to many points and from many points to many points, it is quite difficult to show all the lengths.

Map 3 Results
(Multiple Source to Multiple Source)

	Duration	Length
Dijkstra	9835ms	-
Bellman-Ford	24887ms	-
A Star	37651ms	-
Floyd-Warshall	377ms	-

Figure 14. Calculated results of Map 3

The "Map 4" in Figure 9 contains negative values for costs. Dijkstra, Floyd-Warshall, and A* algorithms have yielded efficient results since other maps do not have negative weights. The results obtained after selecting "Map 4" and calculating a short path from one point to another on this map by all algorithms are shown in Figure 15.

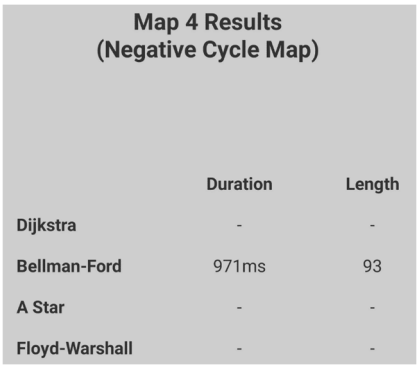

Figure 15. Calculated results of Map 4

3.5. Discussion of the Simulation Results

The results of all the executed algorithms are summarized in Table 1. Here the "Duration" column is the runtime of the algorithm. The "Length" column shows the length of the shortest path that the algorithm can find.

Table 1. Results of the shortest path algorithms (*D*: Duration; *L*: Length)

Algorithm	Map 1		Map 2		Map 3		Map 4	
	D	*L*	*D*	*L*	*D*	*L*	*D*	*L*
Dijkstra	154	377	127	-	9835	-	-	-
Bellman-Ford	10180	377	14144	-	24887	-	971	93
A*	78	377	18703	-	37651	-	-	-
Floyd-Warshall	61518	377	61105	-	377	-	-	-

In the case of Map 1, when the algorithms are run one by one on a map calculating from one point to another in order to find the shortest path, the Dijkstra algorithm found 154 ms, Bellman-Ford algorithm found 10180 ms, A* algorithm found 78 ms, and Floyd-Warshall algorithm found 61518 ms. It is seen from the results that when calculation from a single source to a single source is required, the most efficient algorithm is the A* algorithm. Since the A* algorithm is an algorithm that focuses only on one target, it has found the fastest result by proceeding intuitively unlike other algorithms.

Especially in terms of speed, the A* algorithm is used in strategy games. This is due to the fact that in such games the general provisions require point-to-point calculations. Since the A* algorithm produces intuitive predictive results, it can find the shortest path on the map in a few steps more than other algorithms require. However, this redundancy is reduced so that it goes unnoticed in games by changing the weights used in the algorithm.

In the case of Map 2, when the calculations are made from one point to many points, it is found that the A* algorithm suffers a significant performance loss. This is due to the fact that, by design, this algorithm has to perform computations from every point to every point. The Dijkstra algorithm has yielded a remarkably fast result, and it is seen that the best result is the Dijkstra algorithm when the calculation is made from one point to several points. This algorithm finds the shortest paths from one point to all other points while performing a calculation by its own design. Since it does not calculate separately for all points such as the A* algorithm, it is seen that the speed is much faster. Because of this feature, the Dijkstra algorithm is generally used in navigation applications and network routing protocols that do not contain negative values.

In the case of Map 3, when calculations are made from many points to many points at the same time on this map, Dijkstra and A* algorithms are found to fall behind the Floyd-Warshall algorithm in this calculation. As seen from the obtained results, the Floyd-Warshall algorithm is slower in calculating from one point to another than other algorithms because it tries to find the shortest path for all points given to the algorithm. However, it can be seen that in the case of calculations from many points to many points, this algorithm finds the shortest paths from all points to all points much faster than other algorithms. It is used in video games of tower defense type, where all objects need to know the location of other objects at the same time. The Floyd-Warshall algorithm is also used in more complex network routing protocols as well as Dijkstra.

As seen in Table 1, the "Length" columns for "Map 2" and "Map 3" are empty. Since the calculations made on these maps are obtained from one point to many points and from many points to many points, it is quite difficult to show all the lengths.

The "Map 4" in Figure 9 contains negative values for costs. As seen in Table 1, in the case of Map 4, only the Bellman-Ford algorithm found a result as 971 ms. This is due to the fact that other algorithms do not support negative costs. Dijkstra, Floyd-Warshall, and A* algorithms give efficient results when maps do not have negative weights. Bellman-Ford algorithm is frequently used in graphs containing negative values and Router Information Protocol applications.

Since the Dijkstra algorithm tries to make calculations from one point to all points with a greedy approach, it has been seen that it performs more operations on efficiency than the A* algorithm. It was found that the number of iterations increased because the algorithm controls each note more than the A* algorithm. More iterations mean more working time. Since the A* algorithm is focused on the endpoint, it is found that it is more efficient due to the operations at the closest points to the endpoint without spending time at each point.

Dijkstra algorithm is more controllable than the A* algorithm. Due to the intuitive approach of the A* algorithm, it has been found that the A* algorithm does not always find the optimal path. This is because the A* algorithm performs heuristic calculations based on the last point. In other words, when the number of obstacles on the map increases, it is seen that the A* algorithm thinks that the algorithm will end as it approaches the endpoint. As a result, it ignores the obstacles because it uses an intuitive approach while performing calculations, and this causes a decrease in the performance.

The most important factor that separates the Bellman-Ford algorithm from the Dijkstra algorithm is the negative weights that can be given to the roads on a map. When the negative weight is given to the Dijkstra algorithm, there is the possibility of entering infinite loops or making inaccurate results while performing calculations. This algorithm performs calculations from one point to all other points just like the Dijkstra algorithm. However, it has been found that the Dijkstra algorithm works better if there is no negative weight in the graph used.

The feature that distinguishes the Floyd Warshall algorithm from other algorithms is that it can find the shortest path from any point to all other points. When the same process is performed with other algorithms, it is seen that they do not perform as well as the Floyd Warshall algorithm. For this reason, the Floyd Warshall algorithm is more effective in tower defense games in which each object must know the shortest path between each other.

4. CONCLUSION

In this period of rapid development of technology, people's need for speed is increasing. Depending on how complex and fast the algorithms used, the yields that people can get from these algorithms vary. In this study, the most popular shortest pathfinding algorithms such as Dijkstra algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm, and A * algorithm, used in video games, map calculations, and navigation applications are examined.

In this study, various maps were designed using the application developed for Android, and the performance of the algorithms listed above on these various maps was examined and based on these performances, it was suggested which types of algorithms can be used in which types of games.

Once the user has decided for which purpose to choose the shortest path, he must choose according to this information the most appropriate algorithm taking into account the advantages and disadvantages described above.

As can be seen from the results, no algorithm is determined as the best algorithm. Depending on the purpose of use of the algorithm, it has been seen that the algorithms perform more effectively on certain maps and their advantages are determined. If these results are taken into account and the shortest path algorithm is selected, the memory requirement in the system where the algorithm will be used can be minimized and the algorithm can be run more efficiently to save time.

NOMENCLATURE

GA	Genetic Algorithm
GPU	Graphics Processing Unit
SDN	Software-Defined Networking
SSSP	Single-Source Shortest Path

REFERENCES

[1] H. Bodlaender, "Complexity of Path-Forming Games", Theoretical Computer Science, Vol. 110, Issue 1, pp. 215-245, 1993.

[2] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerical Mathematics, Vol. 1, pp. 269-271, 1959.

[3] M. Sniedovich, "Dijkstra's Algorithm Revisited: The Dynamic Programming Connexion", Control and Cybernetics, Vol. 35, Issue 3, pp. 599-620, 2010.

[4] H. Wang, Y. Yu, Q. Yuan, "Application of Dijkstra Algorithm in Robot Path-Planning", Second International Conference on Mechanic Automation and Control Engineering, pp. 1067-1069, Hohhot, China, 2011.

[5] A. Qaiser, H. Qasim, Z. Tehseen, M. Arfan, "Reduced Solution Set Shortest Path Problem: Capton Algorithm with Special Reference to Dijkstra's Algorithm", Malaysian Journal of Computer Science, Vol. 31, Issue 3, pp. 175-187, 2018.

[6] H. Wang, "Urban Integrated Intelligent Parking Guidance Based on Dijkstra Algorithm", Advances in Transportation Studies, Special Issue 1, pp. 153-164, 2021.

[7] F. Busato, N. Bombieri, "An Efficient Implementation of the Bellman-Ford Algorithm for Kepler GPU Architectures", IEEE Transactions on Parallel and Distributed Systems, Vol. 27, Issue 8, pp. 2222-2233, 2016.

[8] S. Broum, M. Talea, A. Bakali, F. Smarandache, S.K. Patro, "On the Neutrosophic Counterpart of Bellman-Ford Algorithm", International Conference on Advanced Intelligent Systems for Sustainable Development (AI2SD 2019), pp. 107-114, 2020.

[9] K.P.K. Rao, T.S. Murugan, "An Efficient Routing Algorithm for Software Defined Networking Using Bellman Ford Algorithm", International Journal of Online and Biomedical Engineering, Vol. 15, Issue, 14, pp. 87-95, 2019.

[10] Z. Ramadhan, A.P.U. Siahaan, M. Mesran, "Prim and Floyd-Warshall Comparative Algorithms in Shortest Path Problem", Joint Workshop KO2PI and the 1st International Conference on Advance and Scientific Innovation, Medan, Indonesia, April 2018.

[11] S. Hougardy, "The Floyd-Warshall Algorithm on Graphs with Negative Cycles", Information Processing Letters, Vol. 110, Issue 8-9, pp. 279-281, 2010.

[12] S. Solichudin, A. Triwiyatno, M. Riyadi, "Conflict-Free Dynamic Route Multi-AGV Using Dijkstra Floyd-Warshall Hybrid Algorithm with Time Windows", International Journal of Electrical and Computer Engineering, Part I, Vol. 10, Issue 4, pp. 3596-3604, 2020.

[13] J.R. Stuart, P. Norvig, "Artificial Intelligence a Modern Approach", Pearson Education Inc, 4th Edition, New Jersy, USA, 2018.

[14] K. Su, C.Y. Chung, J.T. Zou, T.Y. Hsu, "A* Search Algorithm Applied to a Chinese Chess Game", Artificial Life and Robotics, Vol. 16, pp. 132-136, 2011.

[15] Google Announces on Android, www.theverge.com

/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users.
[16] Industry Leaders Announce Open Platform for Mobile Devices. www.openhandsetalliance.com/press_ 110507.html.
[17] Object-Oriented Programming (OOP) Basics. www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.htm.
[18] A.A. Allahverdiyev, "Methods and Models of Optimization of Transport Logistics", International Journal on Technical and Physical Problems of Engineering (IJTPE), Issue 27, Vol. 8, No.2, pp. 32-39, June 2016.
[19] N.M. Tabatabaei, A. Jafari, N.S. Bushehri, K. Dursun, "Genetic Algorithm Application in Economic Load Distribution", International Journal on Technical and Physical Problems of Engineering (IJTPE), Issue 17, Vol. 5, No. 4, pp. 88-93, December 2013.
[20] S. Balasooriya, I. Kavalchuk, E. Dimla, "Innovative Path Planning Algorithm for an Autonomous Robot with Low Computational Cost", International Conference on Advanced Computing and Applications (ACOMP), pp. 152-157, Nha Trang, Vietnam, 2019.
[21] Dune 2000, http://web.archive.org/web/20000930035519/http://www.westwood.com/games/dune2000/press.html.
[22] Age of_Empires 2, https://en.wikipedia.org/wiki/Age_of_Empires_II.
[23] Shortest path, https://dl.acm.org/citation.cfm?doid=367766.368168.
[24] Algorithms, http://faculty.ycp.edu/~dbabcock/PastCourses/cs360/lectures/lecture21.html.

BIOGRAPHIES

Ulviye Hacizade was born in Baku, Azerbaijan, in 1973. In 1995, she graduated from the Computer Engineering Department, Azerbaijan State Pedagogical University (Baku, Azerbaijan) with honor diploma. He received the Ph.D. degree in Process Control from Azerbaijan Technical University (Baku, Azerbaijan), in 2003. From 1995 to 2005, he worked as a Scientific Worker at the Azerbaijan State Petrol Academy. Since 2006, she has been with Department of Computer Engineering, Halic University (Istanbul, Turkey), where she is currently an Assistant Professor. From 2021, she is Head of the Computer Engineering Department of Halic University. Her research interest includes fuzzy logic, genetic algorithms, modeling, adaptive control systems, data mining.

Giray Bal was born in Istanbul, Turkey in 1990. In 2013, he graduated from the Computer Engineering Department, Halic University (Istanbul, Turkey). Throughout his university life, he strengthened his software knowledge with object-oriented programming languages. He started working at Argela Technologies, Turkey as a product development engineer. In 2016, he started his graduate studies in Computer Engineering Department, Halic University with thesis. He graduated from Master degree with thesis in 2018. He worked at Invio Bilisim, Turkey as Mobile Developer, Full Stack Web and Mobile Developer at Miops Technologies, Turkey. Since 2019, he has been working as a software developer in Vektora Information Technologies Industry 3.0 IoT (Internet of Things) projects with the title of Lead Software Engineer.