Journal of Cases on Information Technology
Volume 24 • Issue 3

Towards Intelligent Road Traffic Management Over a Weighted Large Graphs Hybrid Meta-Heuristic-Based Approach

Mohamed Yassine Hayi, Bouira University, Algeria
Zahira Chouiref, Bouira University, Algeria
Hamouma Moumen, University of Batna 2, Algeria
https://orcid.org/0000-0002-1986-7590

ABSTRACT

This paper introduces a new approach of a hybrid meta-heuristics-based optimization technique for decreasing the computation time of the shortest paths algorithm. The problem of finding the shortest paths is a combinatorial optimization problem which has been well studied from various fields. The number of vehicles on the road has increased incredibly. Therefore, traffic management has become a major problem. The authors study the traffic network in large-scale routing problems as a field of application. The meta-heuristic they propose introduces a new hybrid genetic algorithm named IOGA. The problem consists of finding the k optimal paths that minimize a metric such as distance, time, etc. Testing was performed using an exact algorithm and meta-heuristic algorithm on randomly generated network instances. Experimental analyses demonstrate the efficiency of the proposed approach in terms of runtime and quality of the result. Empirical results obtained show that the proposed algorithm outperforms some of the existing techniques in term of the optimal solution in every generation.

KEYWORDS

Djikstra's Algorithm, Genetic Algorithm, K-Shortest Paths, Meta-Heuristic, Optimization, Routing Problem, Weighted Large Graphs

INTRODUCTION

Research in meta-heuristics is well known in the artificial intelligence field and has also been very active during the last decades. When facing complex new optimization problems, it is very natural to use meta-heuristics techniques (Christian et al., 2008). With the massive growth of the Information, big data optimization has become one of the main researches. Furthermore, complex networks such as the transportation, communication, Internet and distribution networks, etc., have increasingly attracted the attention from various fields of science and engineering. Combining the meta-heuristic with other optimization techniques so called hybrid meta-heuristic can provide an advanced flexibility and more efficient behavior when dealing with large-scale and real-world problems (Christian et al., 2008). Various applications such as transport applications require the using of a meta-heuristic shortest

DOI: 10.4018/JCIT.20220701.oa4

path rather than one of the standard algorithms. The heuristic algorithm that finds the shortest path is optimal and significant and must quickly find the best paths. Therefore, we have thought about using hybrid meta-heuristics to solve the routing issue in road networks in our work.

The graph is an important complex network to describe the relationship among a set of interconnected nodes (entities). The shortest path computation is a frequent operation. In these situations, it is often valuable to be able to find the k optimal shortest paths. In the last decade, the problem of finding the shortest paths is a challenging task over large graphs; it is also investicated as one of the most widely-studied combinatorial optimization problems. Many of already existing graph algorithms are not appropriate for graphs of larger sizes.

In our work, we study traffic network in large scale routing problems as a field of application. Finding a suitable route to reaching the destination in time with saving energy has significance in real-world applications. The vertices (nodes) of the graph correspond to spatial units (cities) of the road network. Edges connect vertices, and they are weighted with non-negative weights according to the distance between vertices. Operations on a graph can take a long time due to the complexity of structural connectivity and the graph's size. Also, a large graph makes efficiency even more difficult (Hosseinabadi et al., 2018). Hence, in this paper, we aim to find the shortest route starting from the source to the destination where the distances between each city are known.

Because the K- shortest path is NP-hard, we combine exact and heuristic approaches. The optimal shortest path algorithms such as Dijkstra tend to be extremely computationally intensive for real-time applications in practical traffic networks. For this reason, we develop a new algorithm combining the exact algorithm (namely Dijkstra's algorithm) with a meta-heuristic algorithm (namely Genetic Algorithm - GA) into a shortest path search process. In this study, the new hybrid algorithm, namely IOGA (Improved Optimization Genetic Algorithm), is developed to find one or k optimal paths containing the minimal cost between two vertices in an oriented and non-oriented graph. So, hybridizing a genetic algorithm with Dijkstra's algorithm is an interesting idea because the classic strategy using Dijkstra's algorithm minimizes the search space whereas the heuristic strategy optimizes the search problem to return the better solutions.

The experimental study makes a comparison of our approximate hybrid method (i.e., IOGA) with three algorithms (Dijkstra's algorithm, Genetic algorithm and A* algorithm) in order to ensure the following specific contributions: i) find one or several shortest paths; and ii) minimize the execution time by comparing with the three algorithms previously mentioned. The effectiveness of our approach that employ the heuristic algorithm is confirmed. The final results illustrate that the proposed approach with the new optimization strategy allowed us to obtain good results with high performance.

The rest of this paper is organized as follows: Some approaches related to the shortest paths problem are briefly introduced in Section 2. In Section 3 we briefly discuss the K-shortest path problem. Section 4 presents an overview of our proposed approach, along with an example for illustration. Section 5 shows the experiments designed to assess the performance of the IOGA. Section 6 provides the results obtained by the experiment. A brief conclusion and future research are finally given in Section 7.

RELATED WORKS

The problem of determining the K-shortest paths has been studied in several works. Due to the issues of numerous and diverse applications, there has been a surge of research in shortest-path algorithms. The computation of shortest-path can generate either exact or approximate solutions by using same algorithms or meta-heuristic algorithms. The heuristic algorithms aim to minimize the computation time. Many works focus on algorithms that target traffic applications, where we summarize some of them here.

Hershberger et al. (Hershberger et al., 2011) reviewed the classical Dijkstra's algorithm and an over road network. And they illustrated heuristic techniques for computing the shortest-path that was

given by a subset of the graph. Goldberg et al. (Goldberg et al., 2014) represented a new algorithm to identify the k shortest simple paths in a directed graph, and they announced its implementation.

Sommer (Sommer, 2013) critiqued chosen approaches, algorithms, and results on shortest-path queries from these fields, with the principal focus reposing on the tradeoff among the index size and the query time. Also, Sommer's survey introduced the transportation network class of algorithms, and he included algorithms for general graphs as well as planar and complex graphs.

Pettie et al. (Pettie et al., 2016) proposed a new algorithm that outperforms Dijkstra's algorithm on sparse graphs for the all-pairs shortest path problem. The presented results show that the new algorithm runs quicker than Dijkstra's algorithm on various sparse graphs when the vertices number varies from a few thousand to a few million. Lee et al. (J. Lee et al., 2012) compared Dijkstra's algorithm and the genetic algorithm and its own DGA algorithm. The results showed that Dijkstra's algorithm does not give the best solutions in the graph, containing more than 10 000 knots; on the other hand, the DGA algorithm gave good results in acceptable execution time.

In (J. Kim et al., 2018) and (Pettie et al., 2016), the authors presented a review of various heuristic shortest-path algorithms. They proposed the main distinguishing features of different heuristic algorithms as well as their computational costs. In (Abraham et al., 2015), the authors attempt to fix the problem of finding an adequate point-to-point shortest path algorithm for graphs of larger sizes by using both the A* algorithm and the genetic algorithm. The bi-directional approach decreases the search space, and the genetic algorithm optimizes the exploration problem to return the best result.

U. Hacizade et al. (Hacizade et al., 2017) used GA based method to find the optimal route for solving the traveling salesman problem for the field audit team in order to speed up the process and reduce the time required. J. Kim et al. (J. Kim et al., 2018) proposed a new algorithm to compute the k shortest simple paths in a directed graph, and they reported on its implementation. The authors prove that GA is used to explore the search space virtually. In this research (Shanmugasundaram et al., 2019), N. Shanmugasundaram et al. applied GA to determine guide path design for one way road network with an objective to minimize the total distance traveled by vehicles within the network. A branch-and-bound technique with the breadth-first search is applied to search the shortest path between the source points and the destination points.

Jamal et al. (Jamal et al., 2020) made a comparison between genetic algorithm and differential evolution (DE) algorithms to enhance the intersection's level of service (LOS) by optimizing the signal timings plan. Simsir et al. (Simsir et al., 2019) came up with a smart solution to find the shortest path for the VRPSDP (vehicle routing problem with simultaneous delivery and pickup) using the Artificial Bee Colony (ABC) algorithm. Ugurlu et al. (Ugurlu et al., 2018) presented a new genetic hybrid algorithm. Also, they proposed a new heuristic algorithm for MWDS (Minimum Weight Dominating Set) to create an initial population. Compared with existing algorithms in the literature, the experimental results show that the hybrid genetic algorithm are faster than these algorithms, they also can yield better solutions than these algorithms.

Broumi et al. (Broumi et al., 2020) have proposed an optimization approach based on a ranking strategy that takes both the length of edges and the number of nodes into account, and this work has a direct relation with our proposed work. Also, Nator Junior et al. (Nator Junior et al., 2020) aim to design actuated traffic controllers optimized by the genetic algorithm; also, Nator Junior et al. (Nator Junior et al., 2020) aim to design actuated traffic controllers using presence sensors and optimized by the genetic algorithm.

Analyzing the works mentioned above, it can be summarized that:

1. Dijkstra's algorithm is one of the best optimization algorithms and it has been successfully applied to a variety of optimization problems and also it is a fast algorithm in the execution time but several researches have confirmed that the performance of Dijkstra's algorithm is not top in the big graph (J. Lee et al., 2012; Singh et al., 2018; Z. Khan, 2016; H. Reddy, 2013).

2. To solving the shortest paths problem, the complexity of an exact algorithm increases with the number of graph vertices.
3. The genetic algorithm is a very well known one and it has a character of biology. It works with the random function, and it is one of the best known heuristic algorithms and it is useful in several areas of optimization.

For these reasons, a genetic algorithm is suggested as the optimal solution to the k-shortest paths problem.

Table 1 shows some comparison criteria of some related works such as: used algorithms, used dataset and type of optimization.

Table 1. Comparison criteria of some related works.

Ref	Year	Algorithm	Dataset	Type of optimization	type of graph
(Changxi Ma et al., 2018)	• 2018	• Genetic Algorithm	• generated by the author (random)	• Optimizing Routing	• Oriented
(Singh et al., 2018)	• 2018	• Dijkstra's Algorithm	• binary map	• Optimal Navigation	• Not mentioned
(Z. Khan, 2016)	• 2016	• Dijkstra's Algorithm • Floyd-Warshall • Bellman-Ford	• generated by the author (random)	• Shortest Path	• Not oriented
(I. Maatouk et al., 2017)	• 2019	• Genetic Algorithm	• generated by the author (random)	• Preventive Maintenance Optimization	• Oriented
(H. Reddy, 2013)	• 2013	• Dijkstra's algorithm • A *	/	• Path finding	• Not oriented
(Duchon et al., 2014)	• 2014	• A *	• Path Planning	• Mobile robot	• Oriented
(Ugurlu et al., 2018)	• 2018	• Genetic Algorithm	• two databases created in random.	• Minimum Weight Dominating Set	• Oriented and Not oriented
(Jamal et al., 2020)	• 2020	• genetic algorithm	/	• Intelligent Intersection Control	• Not oriented
(Simsir et al., 2019)	• 2019	• Artificial Bee Colony algorithm	• benchmark problem data sets	• vehicle routing	• Oriented
(Nator Junior et al., 2020)	• 2020	• genetic algorithm	• traffic signal controller	• Intersection traffic controller	• Oriented
(Broumi et al., 2020)	• 2020	• Neutrosophic Theory	• a numerical example	• Network Optimization	• Not mentioned

Research Highlights

The aims of studying the k shortest paths problem was to determine an appropriate optimal (s) route (s), reaching the destination in both optimal time and distance to maximally reduce total energy consumption. This research highlights four issues:

- We developed an optimal genetic algorithm for solving k-shortest path by using an innovative, hybrid algorithm between the genetic algorithm and Dijkstra's algorithm.
- We created two databases to do our testing between IOGA and both Dijkstra's algorithm, genetic algorithm, and A* algorithm.
- The Both databases can be used in future work by other researchers.
- We assessed the performance of the IOGA in term of the number of solutions obtained, the quality of the solution (minimal solution), and the run time.

THE K-SHORTEST PATHS PROBLEM

The K-shortest paths problem is the base for a lot of combinatorial optimization problems. The problem of finding the shortest path between a starting vertex and a terminal vertex (which exist in a given network diagram) which is widely used in various fields, such as: computer network routing algorithm, Pathfinder robot, navigation route, game design, etc. To formulate a mathematical model to find the K shortest paths between the origin node and the destination node, assuming that G = (N, E, C), a weighted directed graph (G) (we have given an example on a directed graph but we have applied our search on a directed graph and an undirected graph) with a set of vertices (nodes) (N) and a set of directed edges (E) and the edge cost values (C), such as:

- c(s, t): cost of directed edge (s®t) from a source node "s" to a target node "t" (s and t both in N) so that costs are non-negative;
- The returned paths are as short as possible.

The shortest K-path problem is a natural and long-studied generalization of the shortest path problem, in which we seek not one but several paths in increasing order of length. The problem of determining the shortest paths K has proved more difficult. To find the shortest path, we can use the shortest path algorithms. Shortest path algorithms can be classified as either (i) All Even Shortest Path Algorithms (APSP) or (ii) Single Source Shortest Path Algorithms (SSSP). In this paper, we focus on the SSSP algorithms which find the path with the minimal cost from a single origin vertex to all destination vertices in the graph. the Dijkstra's algorithm (E.W. Dijkstra, 1959); Bellman-Ford algorithm (Bellman, 1958; Ford, 1956) and the D'Esopo-Pape algorithm (U. Pape, 1974) are some of the well-established SSSP algorithms.

Dijkstra's algorithm requires the calculation of the shortest paths between the origin nodes and all the nodes closest to the Origin-Destination matrix. Dijkstra's algorithm can solve this problem and it can be extended to find more than one path. The genetic algorithm has excellent performance in searching spaces with large solutions, and can compromise between efficiency and accuracy. For this reason, we tried to solve this problem by applying the Dijkstra's algorithm and the genetic algorithm method presented on section IV. In the next section, we discussed our proposed approach and then we presented the results of our approach with some comparisons to illustrate the performance of our algorithm.

COMPUTING THE K-SHORTEST PATHS

The K-shortest paths problem is solved by implementing a new technique through a heuristic approach in which such a hybrid heuristic can combine the power of classical and heuristic techniques in order to find the optimal solution in a few iterations. We named this algorithm by Improved Optimization Genetic Algorithm (IOGA) or shortly IOGA. In what follows, we first introduced the hybridization principle allowing enumerating the K-shortest paths in a directed and a non-directed graph. Then the new hybrid genetic algorithm was addressed in which the basic steps are customized.

A* algorithm

The A* algorithm is very useful in the field of optimization and is one of the most popular heuristic among all heuristic algorithms due to its completeness, optimality, and optimal efficiency (Russell et al., 2018). One could say that to find a way from one point to another one must start by heading towards the destination. Well, this is exactly the idea that algorithm A* uses. The idea is very simple: at each iteration, we will try to get closer to the destination; we will therefore favor the possibilities that are directly closer to the destination, setting aside all the others.

Hybridization Principle

In the next, we start by describing the hybridization principle of IOGA, and then we presented Dijkstra's algorithm and the genetic algorithm steps.

The IOGA is a hybrid algorithm between the Dijkstra's algorithm and the genetic algorithm. This proposition is a mixture between the two algorithms, from where we worked to build a population, which is a part of GA, from the same principle of the Dijkstra's algorithm. This principle represents the unfolding method of the Dijkstra's algorithm. In this last algorithm, we always calculate the nearest node going from one node to another. On the other hand in IOGA, we calculate the TOP nodes nearest (2, 3, 4,...), so that we will have several proposed routes, we put them in the population. After each iteration, we calculate the distance of each road exists in the population, we leave only the lines which can be an optimal solution, and we suppress the other lines, so that we will not have a single route, but we will obtain several routes of the optimal solution in a quick way.

Dijkstra's Algorithm

Dijkstra's algorithm is the most popular algorithm in the shortest path problem. It is often used in orienting graphs weighted by positive real. It was created in 1959 by the Dutch computer scientist Edsger Dijkstra. This algorithm always searches for the shortest path from one node to all the other nodes. It takes an initial knot, and it searches for the shortest path to all the other nodes, as a second iteration, according to the nodes visited by the first node. It chooses the shortest path of the previous node, and it repeats the first step, and so on until it turns on all the nodes of the graph.

Algorithm 1 represents the Dijkstra's algorithm in detail:

Genetic Algorithm

The GA is a randomized, evolutionary search and optimization algorithm that imitates natural biological evolution (Bhattacharjya, 2012). It has been applied in solving various problems in different human endeavor fields and used to obtain an optimal solution in a reasonable time. It uses a natural selection (random) to have a population. This last will be evolving after each iteration, and when the algorithm reaches a stop condition, we will take the best solution that exists in the population as a final solution for the problem (Hamed, 2010).

Figure 1 illustrates the simplified organigram of a genetic algorithm.

In what follows, we introduced the proposed approach to enumerate the K-shortest paths in a directed graph.

The Proposed Hybrid Meta-heuristic Algorithm

This section is dedicated to introducing the new hybrid genetic algorithm IOGA we propose. This meta-heuristic search strategy and its application in shortest path search are investigated in the following.

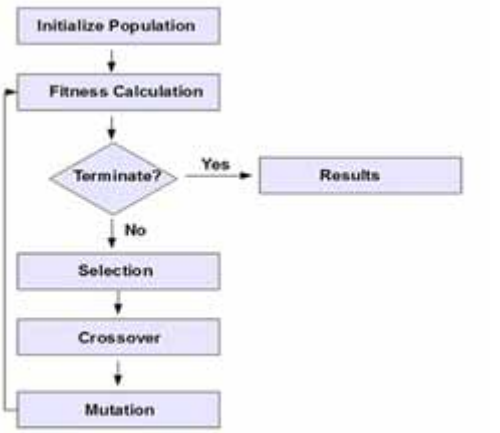
The main phases of the IOGA, as illustrated in Figure 1, are typical of genetic algorithms and are introduced next in detail.

Given a directed graph G with non-negative edge weights and two vertices s and t, the problem asks for the k-shortest paths from s to t in increasing order of length. A positive integer k is returned

Algorithm 1. Dijkstra's algorithm

```
Function Dijkstra (Graph,Source);
Create vertex set Q
For each vertex set Q
Dist[v] = INFINITY
Prev[v]= UNDEFINED
Add v to Q
End for
Dist [source]= Q
While Q is not empty do
U = vertex in Q with min dist[u]
Remove u from Q
For each neighbor v of u:
Alt = dist[u] + length (u,v)
If alt < dist[v]:
Dist [v] = alt
Prev[v]= u
End if
End for
End while
Return (dist[], prev[])
```

Figure 1. The simplified organigram of a genetic algorithm



at the end of the iteration. The structure of IOGA here is resumed in the following basic steps in algorithm 2.

IOGA finds many solutions exists in a logical way, it takes the source of the route, and it adds the TOP (it is a variable) nodes closest to the destination, so that it builds a population of size TOP.

Figure 2 represents an explanatory example on the functioning of IOGA.

The algorithm that we created is very complicated, so to better understand the flow of the algorithm we made a simple example to understand the different stages of our proposal:

- The parameter TOP = 3, i.e. from each node, we will take the closest TOP (3) nodes.
- The parameter SIZE = 4, i.e. in each row of the population we have a source, two knots and a destination.

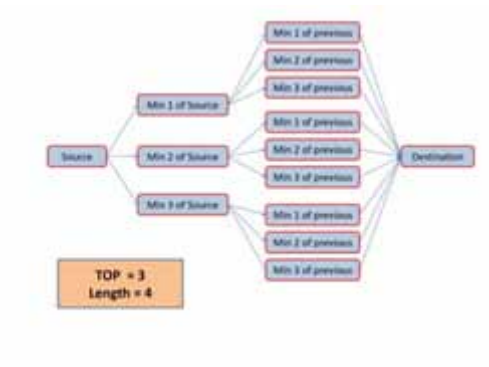
Algorithm 2. Improved Optimization Genetic Algorithm

Inputs:
- Graph (Adjacency matrix)
- Top matrix
- Top
- Size
- Source
- Destination

Outputs:
- the best solution find
- **a list that contains one or more routes that match the solution found**

Add the source to the first population cell
i=2
while (i < SIZE) do
j=1
while (j <= TOP) do
Add the (j) node closest to the previous element in population
j = j + 1
k = 0
end while
end while
while (k < length(population)) do
solution = final_function (population,Destination)
k = k + 1
l = 0
while (l < length(population)) do
population = cleaning (population, solution)
j = j + 1
end while
end while
i = i + 1
Return (population, solution)

Figure 2. An explanatory example on the functioning of IOGA.



We proposed a graph of 10 nodes to make the example. Table 2 represents the adjacency matrix which represents the graph.

For that we propose an example which explains the course of the algorithm, we randomly choose a source (N7) and the destination (N9). We initialize the parameter TOP = 3 and SIZE = 4, and we will run the algorithm according to the SIZE variable (2, 3, 4).

Table 2. Adjacency matrix of the graph.

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
N1	0	15		6	23		19	23		
N2	15	0				16		29		8
N3			0		13			15		
N4	6			0		42	16		26	17
N5	23		13		0	53		19		
N6		16		42	53	0			11	16
N7	19			16			0	5	82	
N8	23	29	15		19		5	0		
N9				26		11	82		0	5
N10		8		17		16			5	0

- SIZE = 2

We check if there is a direct route from the source to the destination. So for this example there is a direct path between N 7 and N 9 with a distance equal to 82.

Figure 3 (SIZE = 2) represents an explanatory diagram of the stage (SIZE = 2) from where the source is in green and the destination in red.

- SIZE = 3

Figure 3 illustrates the explanation of the part of SIZE = 3: the three black balls represent the 3 nodes (8,4,1) closest to the source, after that we check if there is a direct relation between the 3 nodes (8,4,1) and the destination, after the verification we have found a relation only between node 4 and the destination by a value = 26, and we do not have a direct relation between the other two nodes (8,1) and the destination.

So we have only one realizable solution (source, n4, and destination) which has a distance 16 + 26 = 42, this latter solution is less than the previous one (82), so we keep 42 as a better solution until now.

- SIZE = 4

For the part of (SIZE = 4), we have 3 nodes closest to the source and which represent the previous part, and from each node, we will also have 3 nodes, so we will have 9 routes from the source to the destination. We may have one or more blocked roads because sometimes we have no relation between one of the nodes with the destination. Therefore, this road will be ignored.

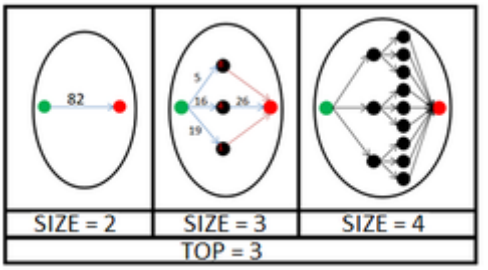
This part is well explained in Figure 3 (SIZE = 4).

The algorithm works by a single population, containing both types of individuals (feasible and unrealizable). The initial population is created using the above algorithm (algorithm 2). A new population is generated from the initial population by performing the operators of selection, replacement, mutation, and crossing of the same algorithm. The population is always ranked in ascending order of physical form.

IOGA contains some part and some function; we explain them in the next lines.

1. Population

Figure 3. The stages of the IOGA algorithm for SIZE = 2, 3 and 4.



It is a matrix which contains several routes, in the genetic algorithm, this matrix is generated randomly, on the other hand in our algorithm, we generate this matrix in a logical way, that is to say according to rules, for example if we take the top parameter equal to 6, so from the source we have 6 first path proposed for the population, the 6 paths starting with the 6 node closest to the destination, and so on, like that we will have a population which has a dynamic size, because after each iteration (size) we delete the paths which do not allow giving optimal solutions by using the Cleaning function (Z. Khan, 2016).

2. Cleaning function

The aim of the cleaning function is avoiding the non optimal solutions or that gives complicated and larger solutions of the solution already found. After the end of each iteration (SIZE) we apply the cleaning function on the population to remove the solutions which can never be less than the MIN solution that we have found so far.

We apply the Cleaning function at the end of each part of size, for example if we have the parameter size which equal to 8, we apply the Cleaning function 7 times, (we do not apply this function in the first phase that contains only the source). The Cleaning function has two objectives:

- Fitness: it is an objective function which calculates the solution of each line of the population in a given moment, and compares this solution with the best one find for now.
- Cleaning: it is to delete each line who exist in the population and which gives a bigger fitness result than the best solution find for now. In general, the Cleaning function works to calculate the solutions of the different routes that exist in the population, and to delete each route which cannot be an optimal solution, to reduce the size of the population and therefore to save time.

Algorithm 3 represents the cleaning function.

3. Final function

The final function has the purpose of detecting the lines that give workable solutions, and of finding the best solution to date. This function traverses all the lines of the population in a given time, and checks if there is a direct path between the last element of the line with the destination, if so we say that the line represents a feasible solution, and compare this solution achievable with our best solution, and of course if the feasible solution is lower than the best solution, we keep the feasible solution as a better solution, and also we save this population line because it represents the path of the best solution to find now. Algorithm 4 represents the final function.

Algorithm 3. Cleaning function

```
def cleaning(population,solution):
while (i<length(population))do
j = 1
s = 0
while (j < length(population[i])) so
s=s+distance(population[i][j-1],population[i][j])
j=j+1
end while
end while
if (s>solution) then
del (population[i])
end if
i=i+1
Return (solution)
```

EXPERIMENTS

In order to verify the performance of the proposed approach, both Dijkstra's algorithm, A* algorithm and IOGA have been implemented in python, and all the experiments have been carried out on Intel (R) Xeon (R) cpu E5 - 2620 v3 2.40 ghz 2.40 ghz (16 GB RAM, 64-bit OS x64 processor). We randomly generated by python two oriented and non-oriented networks of road traffic. We applied our algorithm for two datasets, the first dataset contains 10,000 nodes, and this is a oriented dataset, and the second dataset is a non-oriented dataset which contains 5,000 nodes, each node represents a town. The generation of the two datasets was done according to the work in (Pettie et al., 2016), we generated the two datasets randomly and saved in Excel format, the distance between two nodes is confined between 0 and 9999 (may not have a relationship between two nodes), it means that it is not a complete graph, and also we generated randomly for each graph 500 pairs source/ destination.

Figure 4 represent the graphs of the two databases in three dimensions.

RESULTS

As we mentioned previously, we did the experimentation on two graphs, oriented and non-oriented graphs, and we compared the IOGA algorithm with the Dijkstra algorithm for the oriented graph, then we compared IOGA with Dijkstra's algorithm, genetic algorithm and A * for the non-oriented graph.

Oriented Graph

For the oriented graph we choose to explain the results of the 9 couples that we have chosen randomly, to see in detail the result obtained, and we cannot explain the results of the 500 couples. We notice that Dijkstra's algorithm always gives a single solution, that we name a single route, on the other hand IOGA gives between one up to 5 solutions, and an average of 2.66 solutions for each couple (AVG of 500 source / destination pairs).

For the run time, Dijkstra's algorithm have 1.98 seconds as an average for the 500 couples, and IOGA scored 3.07 seconds, but it recovers several routes for each couple. Table 3 illustrates the number of solutions and the run time in seconds of each algorithm for 9 source/ destination pairs.

We compared the execution time of each solution for the 9 couples. For example, the couple 375 which obtained by the algorithm IOGA 5 solutions in 4.3 seconds, that is to say an average of 0.86 seconds for each solution, knowing that by Dijkstra's algorithm it had only one solution in 2.2 seconds.

Figure 5 represents a graph of comparison of the execution time between the two algorithms IOGA and Dijkstra.

Algorithm 4. Final function

```
def final_function(population,destination,solution):
i=0
while (i<length(population))do
var = length(population[i]) - 1
if (exists_direct_path(population[i][var],destination)) so
s=0
end if
end while
j=0
while (j<var) do
s=s+distance(population[i][j],population[i][j+1])
j=j+1
end while
s=s+distance (population[i][j],destination)
if (s < solution) so
solution=s
end if
i=i+1
Return (solution)
```

Figure 4. Graphs of the two datasets

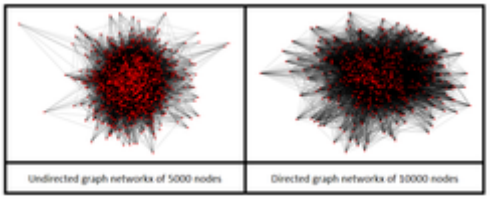


Table 3. Result of oriented graph

	Dijkstra		IOGA	
	Number of solution	Run time (s)	Number of solution	Run time (s)
Couple 375	1	2.2	5	4.3
Couple 163	1	2.6	4	4.1
Couple 29	1	1.8	4	3.8
Couple 407	1	1.6	3	4.2
Couple 149	1	1.8	3	2.9
Couple 380	1	2.3	2	1.7
Couple 293	1	1.7	1	2.3
Couple 73	1	2	4	1.7
Couple 418	1	1.9	1	2.7
AVG	1	1.98	2.66	3.07

Figure 5. Graph representing the execution time for the first 9 couples in oriented graph

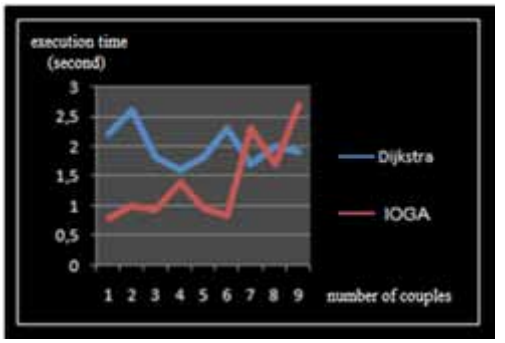


Figure 6 represents the graph of the path number obtained by the IOGA for the first 100 pairs (source/ destination) by the parameters (TOP = 8 and SIZE = 10). We note that the IOGA did not find 4 solutions out of 100 by contribution to the algorithm of Dijkstra's algorithm (73, 82, 93, 96), on the other hand it found the same solutions as Dijkstra's algorithm in 62 cases (one solution), but most importantly, IOGA to find in 34 cases several solutions (from 2 to 8 paths) knowing that Dijkstra's algorithm always returns a single route. The best solution is marked for couple number 87, hence the IOGA returns 8 different paths of the same solution as the Dijkstra's algorithm, and we can have other solutions if we execute the program with higher values of both parameter TOP and SIZE.

Figure 6. Graph representing the number of solutions for the first 100 couples

We have also presented our results in Table 2 which contains the number of solutions per contribution to the parameter of the IOGA (TOP and SIZE).

The results of Table 2 represent the number of path solutions for the first 100 pairs (source/ destination). For the parameter (TOP = 10 and SIZE = 10), we had a case where our algorithm did not give the same result as Dijkstra's algorithm (a result less than that of Dijkstra's algorithm). Also we had 64 cases out of the 100 cases where IOGA had to give only one path of the same solution as Dijkstra's algorithm and most importantly. IOGA returns two solutions in 15 out of 100 cases, knowing that Dijkstra's algorithm only gave one path. Finally for 20 cases, IOGA found several paths (between 3 and 8 paths). We had focused on the example of (SIZE = 10 and TOP = 10) because the execution time of the IOGA does this in almost the same execution time of Dijkstra's algorithm, on the other hand if we used TOP = 35 we will have a great run time.

Non Oriented Graph

For the non-oriented graph, we applied the four algorithms on the non-oriented database.

Table 4. Number of solutions result table

SIZE	15	15	11	10	10
TOP	**35**	**15**	**15**	**10**	**8**
0 solution	0	0	0	1	4
1 solution	64	64	64	64	63
2 solutions	12	12	12	15	18
3 solutions	12	12	12	10	7
4 solutions	4	4	4	3	1
5 solutions	4	4	4	4	4
6 solutions	2	2	2	1	1
7 solutions	1	1	1	1	1
8 solutions	0	0	0	1	1
9 solutions	1	1	1	0	0
AVG	1.88	1.88	1.88	1.77	1.65

The following table represents the results obtained for the first 5 pairs, and the average of the 500 source/ destination pairs.

Table 5 explains briefly the comparison between the first five couples. We recorded good results for the IOGA algorithm, for example for the couple number 2, IOGA had the best result with an average value of 215, and also as an average, with the algorithm IOGA, we obtained the best average of 209.64. With Dijkstra's algorithm we obtained an average value of 211.6, with A* algorithm we had an average value of 213.38 and finally with genetic algorithm we obtained an average value of 217.94.

Table 5. Comparative table between the fourth algorithms on the first 5 couples

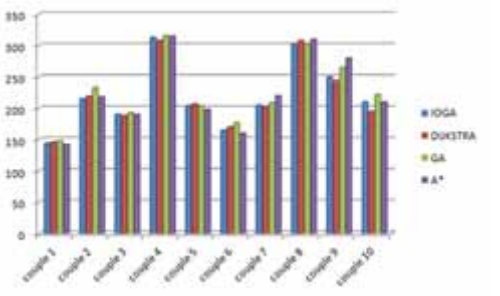
TOP	IOGA	Dijkstra	GA	A*
Couple 1	144	146	148	142
Couple 2	215	219	233	218
Couple 3	190	188	193	190
Couple 4	313	308	316	315
Couple 5	204	207	204	198
AVG	209.64	211.6	217.94	213.38

The following graph represents a comparative histogram between the results of the 4 algorithms for the first 10 pairs.

According to the results of the histogram, we notice that the IOGA algorithm has the smallest result in the majority of cases compared to the other algorithms, in second position the Dijkstra's algorithm has one of good results, but it remains that IOGA gives several solutions and Dijkstra gives a single result, and that is an important point in our proposed algorithm.

Since the end of the 1990s, "complexity" (Steve Maguire et al., 2006) has become a necessary calculation to see the quality and the speed of an algorithm. The goal of complexity is to allow a direct

Figure 7. Comparative histogram between the results of the 4 algorithms for the first 10 pairs



comparison between several algorithms according to the execution time or more precisely according to the number of instructions of each algorithm.

In the following, we will calculate the complexity of each algorithms used in our research, the table 6 represents the complexity of each algorithm.

We notice that Dijkstra's algorithm gives the best complexity of O ((a + n) log n), then IOGA algorithm and Genetic algorithm in second position and by the same complexity O (n * Ln (n)). In the last position, algorithm A * by a strong complexity of O (n2). Therefore, we can conclude that Dijkstra's algorithm has the best complexity and it remains the best known algorithm, but the IOGA algorithm has an advantage of the recovery of several paths by a complexity a little higher than Dijkstra's algorithm.

Table 6. complexity table of the four algorithms

Algorithm	IOGA	Dijkstra	GA	A*
Complexity	O(n * Ln(n))	O((a + n)log n)	O(n * Ln(n))	$O(n^2)$

CONCLUSION AND FUTURE RESEARCH

Heuristic search strategies have the potential to dramatically speed up a shortest path algorithm. The K-shortest paths problem is the base for a lot of combinatorial optimization problems. In this paper, we focused on the calculation of K-shortest paths problem on a road network. The K-shortest paths problem in which we seek a path corresponding to the minimum cost from an origin to a destination node in a network under some constraints.

An enhanced optimization approach based on the integration of the exact algorithm (Dijkstra) and the metaheurstic algorithm (GA) is proposed to improve the performance of shortest paths calculation for the road network. GA is one of the most powerful meta-heuristic methods. Through the use of a hybrid algorithm, the execution time of IOGA is decreased when solving the shortest paths problem.

The IOGA finds one or k paths containing the minimal cost between two vertices in a oriented and non oriented weighted large graph. This algorithm utilizes strategies which have proved to be efficient in solving shortest paths problem. Testing was performed using generated and random networks. We foresee that the proposed algorithm will be very useful for the current intensive studies of real applications of complex networks.

We pursue the following directions in the next researches:

- Apply IOGA on other datasets and compare our results with other approaches.

- Exploring the possibility of extending the various heuristic search strategies to dynamic and/or stochastic networks.
- Application of the IOGA to a real case study of road traffic taking into account various real constraints such as the problem of traffic lights or the detection of works or accidents that block the roads.
- Applying IOGA on the travelling salesman problem or/and Knapsack problem.
- Hybridization of IOGA with other techniques like constraint programming.

REFERENCES

Abraham, G., & Shukla, D. (2015, January). Shortest Path Computation in Large Graphs using Bidirectional Strategy and Genetic Algorithms. *International Journal of Computer Applications, 109*(13).

Bellman. (1958). On a routing problem. *Quart. Appl. Math.,* 87-90.

Bhattacharjya. (2012). *Introduction to genetic algorithms.* Department of Civil Engineering, Indian Institute of Technology Guwahati.

Blum, Aguilera, Roli, & Samples. (2008). *Hybrid metaheuristics an emerging approach to optimization.* Springer.

Broumi. (2020). *Finding the Shortest Path With Neutrosophic Theory: A Tool for Network Optimization.* Faculty of Science Ben M'Sik, University Hassan II, Morocco. .10.4018/978-1-7998-1313-2.ch001

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Nume. Math.*, *1*(1), 269–271. doi:10.1007/BF01386390

Duchon. (2014). *Path planning with modified A star algorithm for a mobile robot.* Modelling of Mechanical and Mechatronic Systems MMaMS.

Goldberg, H. B., Delling, D., Müller-Hannemann, M., Pajor, T., & Sanders, P. (2014). Route Planning in Transportation Networks. Academic Press.

Hacizade & Kaya. (2017). GA Based Traveling Salesman Problem Solution and its Application to Transport Routes Optimization. *IFAC Papers Online, 51*(30), 620–625.

Hamed. (2010). A genetic algorithm for finding the k shortest paths in a network. *Egyptian Informatics Journal, 11,* 75–79.

Hershberger, Maxely, & Suri. (2011). *Finding the k Shortest Simple Paths: A New Algorithm and its Implementation*. Academic Press.

Hosseinabadi & Vahidi. (2018). *Extended Genetic Algorithm for solving open-shop scheduling problem.* Springer-Verlag GmbH Germany.

Jamal, . (2020). *Intelligent Intersection Control for Delay Optimization: Using Meta-Heuristic Search Algorithms*. King Fahd University of Petroleum and Minerals.

Jr, F. (1956). *Network flow theory. Tech. rep*. DTIC Document.

Junior. (2020). An Intersection Traffic Signal Controller Optimized by a Genetic Algorithm. *International Journal of Computer Applications, 176*(4).

Khan, Z. (2016). *Comparison of Dijkstra's Algorithm with other proposed algorithms*. Virtual University of Pakistan. doi:10.13140/RG.2.2.22743.88480

Kim & Kyun. (2018). *Genetic Algorithms for Solving Shortest Path Problem in Maze-Type Network with Precedence Constraints.* Springer Science+Business Media, LLC.

Lee, J., & Yang, J. (2012). *A Fast and Scalable Re-routing Algorithm based on Shortest Path and Genetic Algorithms* (Vol. 7). Seoul National University Research Park.

Ma, He, & Zhang. (2018, August). *Path optimization of taxi carpooling.* School of Traffic and Transportation, Lanzhou Jiaotong University, Lanzhou, China.

Maatouk, I., Jarkass, E., Châtelet, E., & Chebbo, N. (2019). Preventive Maintenance Optimization and Comparison of Genetic Algorithm Models in a Series–Parallel Multi-State System. *J. Intell. Syst.*, *28*(2), 219–230. doi:10.1515/jisys-2017-0096

Maguire. (2006). *Complexity science and organization studies*. McGill University.

Pape, U. (1974). Implementation and efficiency of Moore-algorithms for the shortest route problem. *Mathematical Programming*, *7*(1), 212–222. doi:10.1007/BF01585517

Pettie, R., & Sridhar, S. (2016). *Experimental Evaluation of a New Shortest Path Algorithm*. Department of Computer Sciences The University of Texas at Austin Austin.

Reddy. (2013). *Path Finding - Dijkstra's and A* Algorithm's*. Academic Press.

Russell, S. J. (2018). *Artificial intelligence a modern approach* (4th ed.). Boston: Pearson.

Shanmugasundaram, K. S. (2019). Genetic algorithm-based road network design for optimising the vehicle travel distance. *Int. J. Vehicle Information and Communication System, 4*(4).

Simsir & Dursun. (2019). *A metaheuristic solution approach to capacitied vehicle routing and network optimization*. Karabuk University, Computer Engineering Department.

Singh, Sutton, & Hatton. (2018). Towards use of Dijkstra Algorithm for Optimal Navigation of an Unmanned Surface Vehicle in a Real-Time Marine Environment with results from Artificial Potential Field. Autonomous Marine Systems Group, Plymouth University.

Sommer. (2013). *Shortest-Path Queries in Static Networks. ACM Computing Surveys*.

Ugurlu & Tanir. (2018). *A Hybrid Genetic Algorithm for Minimum Weight Dominating Set Problem*. İzmir Bakırçay University.

Hayi Mohamed Yassine is a doctoral student at the University of Bouira (specialty artificial intelligence), and also a temporary teacher at the University of Bouira and also in the University of Médéa, he has an MSC Master of Science degree in computer science. He participated in an international conference in 2020 at the University of Oran.

Zahira Chouiref-Latreche is a lecturer and researcher in the Computer Science Department at Bouira university-Algeria. She has been awarded an honorary Doctorate in Computer Science at USTHB-Algiers in 2017. She received his magister in Software Specification and Information Processing at UMBB university-Algeria in 2008. Zahira Chouiref is graduated from UMBB-Algeria university in 2005 with a bachelor's of nature and life science specialty in 2000 from Abderhmane Mira High school. She has more than fifteen years of professional experience in scientific research. She has published papers in the important international conferences and a paper in a journal with high impact factor. Her research interests include computational intelligence techniques in decision making, machine learning, recommendation systems and user-centered personalization, data analytics in the Internet-of-things and fuzzy logic.

Hamouma Moumen is an Associate Professor at the University of Batna 2. He is the Head of computer science department. His main research interests are the basic principles of distributed computing systems. He has published papers in the IEEE NCA, OPODIS, PODC, and ICDCN international conferences and a paper in the Journal of the ACM. He is the recipient of "the Best Paper Award" at ACM PODC 2014.