A new metaheuristic inspired by gravity waves (MIGW) for finding the shortest path in an urban road network.

Dr. Cheikh MOUILAH [a*], Dr. Mahmoud FAHSI [b]

[a] Biomathematics Laboratory, Univ. Sidi Bel-Abbes, P.B. 89, 22000, Algeria

[b] Evolutionary Engeneering & Distributed Information Systems Laboratory , Univ. Sidi Bel-Abbes, P.B. 89, 22000, Algeria

Abstract: The shortest path problem in a graph is a concern of a lot of research since the fifties of the last century, in this paper we try to solve this problem with a new metaheuristic inspired by the confrontations of two gravity waves on a liquid surface. The starting point (source) is the center of the first wave, while the destination point (destination) is the center of the second wave. To find the shortest path, this metaheuristic uses four populations which progress in parallel on the urban network, the meetings between its individuals give several optimal routes in both directions (source—destination and destination—source). The parallelism of the progression of these populations and the mutual competition of threads on common resources are the subject of this paper.

Keywords: Shortest path, urban road network, metaheuristics, evolutionary algorithm, Multithreaded, gravity waves.

1. Introduction

For route calculation in the road network, the literature search shows that there are many shortest path calculation algorithms, and several classifications of these algorithms are proposed, which vary depending on the type of graph used (static or dynamic) or the type of its weighting (deterministic or stochastic).

The first algorithm that finds a minimum spanning tree [1] is published by Prim in 1957, two years later, the “Bellman-Ford-Moore” shortest path algorithm appeared, it is based on Ford’s centralized solution [2] and Bellman’s system of equations [3]. And in the same year, the first Dijkstra algorithm was published to solve the shortest path problem with positive weighting [4]. The A* Algorithm [5] is another deterministic and fast algorithm (if a good heuristic function is available) used to find the shortest path in a graph. The A* algorithm is a good compromise between computational cost and optimality of the solution. Hub Labeling—is considered in several research—is the fastest algorithm in this domain [6].

The use of these exact algorithms is limited to problems of small sizes, but in gigantic graphs these algorithms lose their efficiency and their speed. The metaheuristics are in particular the evolutionary algorithms (genetic algorithms [GA] [7] [8] [9] [10] [11] [12], Ant Colony Optimization with its variants [ACO] [13] [14] [15] [16] [17], the optimization by particle swarms [OSP] [18] [19] [20] [21], the hybrid algorithms [22] [23] [24] [25] and the algorithm inspired by the behaviors of living animals [26]), working on populations of solutions, are well adapted to the optimization in this type of graph.

The objective of this research is to develop a decision support system for the choice of the shortest path in an urban transport system. To do so, we will implement a new optimization metaheuristic inspired by a natural behavior observed on liquid surfaces. The simultaneous fall of two solid masses on these surfaces causes opposite waves called “gravity waves”. These waves progress and grow until they collide. The exploitation of this phenomenon to find the optimal routes in urban transport network, appeared an interesting solution and it constitutes the center of interest of this paper. This solution will be known in this paper as: Metaheuristics Inspired by Gravity Waves (MIGW).

2. Metaheuristics inspired by gravity waves (MIGW):

The principle of this metaheuristic is to launch two waves simultaneously in an urban road graph as on a surface of a liquid, the confrontation of these waves gives routes between the center of the first wave (source) and the second wave (destination). To do this, we need four different populations (P1, P2, P3 and P4). The first two populations P1 and P2 search for the route from the source to the destination, while P3 and P4 find the opposite route (from the destination to the source).

Each population is composed of a set of individuals who reproduce, die and inherit their experiences (their routes) to their descendents.

The principle of this metaheuristic is to place an individual (the root) of each population in the appropriate node (the start node or the destination node), then these individuals, by exploring the graph, develop by building their populations. The meeting between the individuals of these different populations causes three possible events, which can be summarized in this mathematical model (Eq: 1):

Let :

x and y : are two individuals that belong to two different populations.

R: The function of the meeting between two populations constituting the same route.

$$\forall\, x \in \{P1\}\ and\ \forall\, y \in\ \{P2\};\ \ R(x,y) =\ the\ route\ between\ x, y\ is\ found$$
$$or \tag{1}$$
$$\forall\, x \in \{\mathrm{P3}\}\ and\ \forall\, y \in\ \{\mathrm{P4}\};\ \ R(x,y) =\ the\ route\ between\ y, x\ is\ found$$

On the other hand, the meeting between two individuals of the same population *P1*, *P2* causes a circuit, then the individual causing this situation must be eliminated (Eq: 2).

$$\forall\, E\ \in\ \{P1, P2, P3, P4\};\ \forall\, P_1, P_2 \in E;\ \ R(P_1, P_2) =\ circuit\ found \tag{2}$$

A population is created by a single individual placed in a node (source or destination). The individual reproduces in each intersection of the streets, the number of reproduced children depends on the number of adjacent roads of the current street of the father.

Let, P is an individual of any population, the reproduction of P on the street *i, j* and during the iteration t is modeled as follows (Eq: 3):

$$P_{ij}\,(t) = \begin{cases} Kill\ this\ person & if\ n = 0 \\ Create\ (n\ Person\ child)\ and\ Update(T) & if\ n > 0 \end{cases} \tag{3}$$

Hence :

P: An Individual from a Population;

n: The number of adjacent streets of the street *i, j*;

T: The genealogical table.

The algorithm uses two memories during the reproduction of the individuals and the search for the solutions and which are:

- The genealogical table: As its name indicates, it keeps the ascending or descending genealogy of an individual;
- The local table: each individual has his own memory which keeps the paths of his ancestors and his route. This table will be inherited by his sons in his reproductions.

The development of the solutions is based on the traceability of the routes of the individuals in the road network, the algorithm so uses the genealogical table to know the roots of the individuals who found the solution, and the marking of the streets to know their itineraries.

Inspired by the marking behavior of living animals, each individual marks the streets he has traveled and save his footprint (identifiers) in a specific rubric called the marking rubric. The latter is reserved for a unique population. So, each street of the network has four marking rubrics which will be used as a means of communication between individuals and as a method for developing solutions.

Once the street *i, j* is walked by the individual *P*, he stores his identifier in the marking field of this street M_{ij}, following the mathematical model (Eq: 4):

$$P_{ij}\,(t) = \begin{cases} circuit\ found\,; & if\ M_{ij}\ \neq 0\ and\ \ M_{ij}\ = \mathrm{P_{sp}} \\ solution\ found\,; & if\ M_{ij}\ \neq 0\ and\ \ M_{ij}\ = \mathrm{P_{ap}} \\ M_{ij}\ = Person\,; & if\ \ M_{ij}\ = 0 \end{cases} \tag{4}$$

Hence:

$P_{ij}\,(t)$: Individual P in street i j, during iteration t;

P_{sp} : Person from the same population;

P_{ap} : Person from another population;

M_{ij} : The marking rubric of the current street i, j.

The method of progression of these four populations is a decisive factor in the quality of the solution found or in the time needed to find it. The MIGW algorithm uses the parallel evolution of populations; this technique perfectly exploits the physical and software resources available to have a correct, optimal and very fast solution. Its principle is to launch all the populations to progress simultaneously, each thread takes care of a population, the communication between these threads is ensured by the shared variables.

As soon as two individuals belonging to two competing populations meet, a solution is obtained and the path between the source and the population is composed of the routes of these two individuals. We can also exploit the paths of the latter to develop other solutions in order to choose the optimum between them. The principle of this research is based on the genealogical table and the marking of the streets, the adequate mathematical model of this situation is as follows:

Let, R is the route between the source and destination, the composition of this route R during the iteration t is modeled as follows (Eq: 5):

$$R_{ij}(t) = \begin{cases} End & i = source\ or\ j = Destination \\ SelectedWay = way(j,k) & M_{ij} = P \text{ and } M_{jk} = T(P) \end{cases} \quad (5)$$

Hence:

P : An individual from a population;

T : The genealogical table and *T(P)* is the father individual of *P*;

M_{ij} : The marking rubric of the current street i, j;

M_{jk} : The marking rubric of the street *j,k* adjacent to the street *i,j* ;

Once the vertex of the current street end equals the source or destination node then the search for this route is complete and the route is composed.

3. Metaheuristics inspired by gravity waves (MIGW) :

Parallel programming is very requested in the domain of optimization especially to speed up the execution time of algorithms of optimization problems. Several programming languages offer tools for powerful parallelism to exploit this technique.

Multithreading is a technique often used to exploit parallelism by using all the cores of a processor.

Each population of our algorithm uses a thread to discover the road network, which is common (shared) between all populations. Each thread calls its own function to progress its population independently of the other competing threads (Fig: 1).

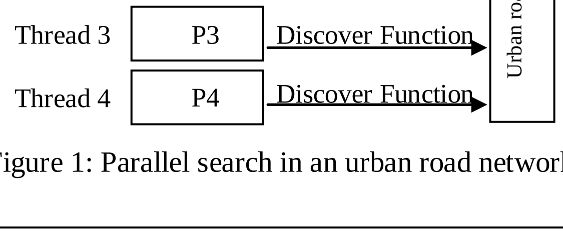

Figure 1: Parallel search in an urban road network.

The idea of having the road network shared between several threads. This automatically causes mutual concurrency between the threads, either in reading the streets, or in marking them. The "mutex. h" library of Microsoft Visual Studio c++ is a good tool to solve the conflicts caused by this competition of threads. This library lists all the access requests to the shared variables in The waiting line to allow only one access at a time during any reading or writing (fig: 2).

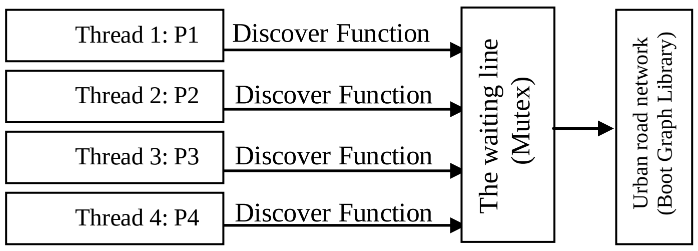

Figure 2: Mutual competition and the Mutex.

Shared variables: two common variables are used in the algorithm by all threads:

The urban road network graph is considered as a search and marking area for all competing populations.

The table of solutions: This table stores the solutions found by the individuals of each population. It will then be used to find the optimal route among all the existing solutions.

The stop variables: is a communication variable, it is used between two threads of the same route. If a thread discovers that its population is empty or is no longer progressing, this thread informs its concurrent that it must abandon the search since it is useless.

4. Pseudo code of the proposed algorithm (MIGW):

```
#Include « mutex.h»;
void Evolution(G &graph, int source, int target)
{
/*********************** création du thread ********************************/
    thread *threads[THREAD_COUNT];
std::vector<Solutions> Solutions;
bool AbordAdvance = false, AbordBack = false;
   /***************** creation of threads ********************************/
    threads[0] = new boost::thread(Discover<graph_t>, &graph, 0, source, target, &Solutions, &AbordAdvance,
&AbordBack);
threads[1] = new boost::thread(Discover<graph_t>, &graph, 1, target, source, &Solutions, &AbordAdvance,
&AbordBack);
threads[2] = new boost::thread(Discover<graph_t>, &graph, 2, source, target, &Solutions, &AbordAdvance,
&AbordBack);
threads[3] = new boost::thread(Discover<graph_t>, &graph, 3, target, source, &Solutions, &AbordAdvance,
&AbordBack);
    /****************** Cleanup *********************************/
for(int i = 0; i < THREAD_COUNT; i++)
{     threads[i]->join();
      delete threads[i]; }
}}
void Discover(G *graph, int Thread, int source, int Target, std::vector<Solutions> *Solutions, bool *AbordAdvance,
bool *AbordBack)
{
    for (int i=0 ; i < (int) thisGeneration.size(); i++)
{      /****************** Get adjacent streets to source *******************/
 thisPerson = thisGeneration[j];
     adjacent_Edges = Get_adjacent_Edges(Thread,source);
     for (int j=0 ; j < (int) adjacent_Edges.size(); j++)
   {   edge_descriptor thisRoad = adjacent_Edges[j];
        if ((boost::target(thisRoad,(*graph))==Target && (Thread==0||Thread==2))||
           (boost::source(thisRoad,(*graph))==Target && (Thread==1||Thread==3)))
              /// a person arrives at the destination, creating a new solution
```

```
                  CreateNewSolution(Solutions, Thread, thisRoad); break;
   else  {
            a.lock();
           int MaketThisRoad = LoadMaketThisRoad<graph_t>(graph, thisRoad, Thread);
           a.unlock();
            if (MaketThisRoad == CircuitExist)
                /// a circuit is found, killing this person
                      thisPerson->Kill = true;
            else if (MaketThisRoad == SolutionExist) {
                /// a solution is found, creating a new solution
                CreateNewSolution(Solutions, Thread, thisRoad);
                thisPerson->Kill = true; break;     }
            Else if (MaketThisRoad == EmptyExist) {
               /// this road is not marked, creating a new individual
 Persons* Child = CreateNewPerson(thisGeneration, thisRoad);
             SetMaketThisRoad<graph_t>(graph, thisRoad, Child->id, Thread);
Update_genealogical_table(genealogy_table, thisPerson->id, Child->id);
                                      }
            }}}}
```

5. Results:

Inspired by the work of PgRouting, the algorithm’s MIGW is programed in Microsoft Visual Studio 2010 C++, then it is converted into a dynamic library (DLL) and added to the Postgresql extension. The experiments are applied on real data of three large Algeria Cities (Algiers, Oran, Sidi Bel Abbes). These data are extracted from the project www.openstreetmap.com and then inserted in the geographic database Postgresql/PostGis.

The evaluation of this proposed solution (MIGW) depends on several important factors that can be summarized as the following questions

- Is the algorithm sensitive to the size of the urban road network graph?
- Are they obtained solutions correct and optimal?
- Does the algorithm require enough time to obtain solutions?

The algorithm is applied on three urban road graphs of three Algerian cities [Algiers, Oran, Sidi Bel Abbes]. Each graph of these cities is composed of edges (the streets of the city) and vertices (the intersections of these streets) (Tab: 1):

Cities	**Number of edges**	**Number of vertices**
Algiers	**38715**	**29921**
Oran	**20442**	**13577**
Sidi bel abbes	**7581**	**5113**

Table 1: The creation of the topology of the three experimental cities.

To evaluate this algorithm, we created three sets of 1000 randomly selected routes [the rand function of c++ is used]. Each set is reserved for a single city.

The results obtained prove that the MIGW’s algorithm is not sensitive to the size of the graph, nevertheless, its execution time depends directly on the length of the route found. In effect, this dependence is completely justifiable, since “the longer the route the bigger the search area”. The time required to have a route of 143 segments, the MIGW algorithm takes fewer than 40 milliseconds (Fig:3):

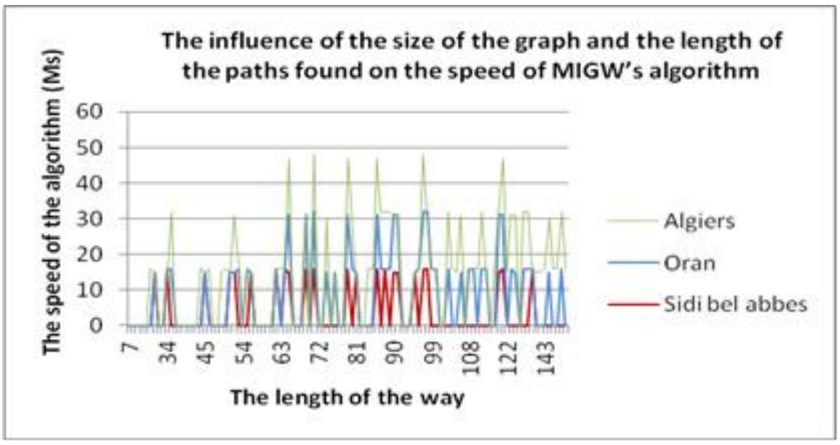

Figure 3: The influence of the route length on the execution time of the algorithm.

The experiments in the figure below (Fig: 4): confirmed the idea that Dijkstra's algorithm is sensitive to the size of the road network. To find the same length of a route in different cities, the algorithm takes three different times (Fig: 4):

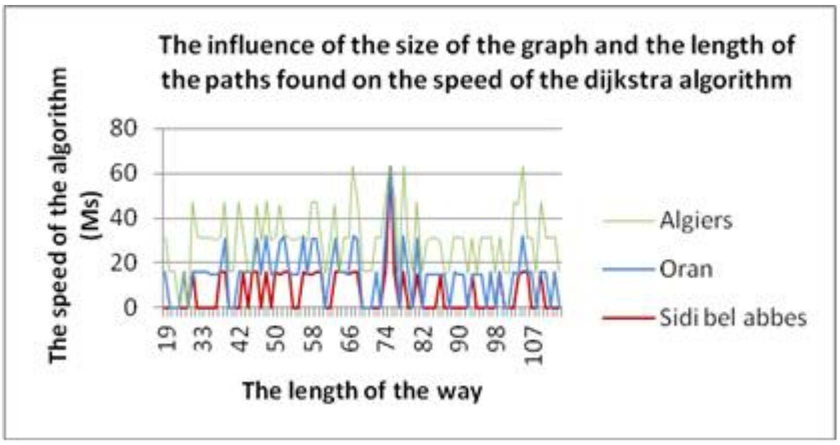

Figure 4: The influence of the route length on the execution time of the algorithm.

In the (Fig: 5) we tested the speed of the two algorithms [MIGW and disjkstra], the results obtained are as follows:

- The MIGW algorithm was ahead of the Diskstra algorithm in the three different cities [Sidi Bel Abbes, the MIGW algorithm is faster 398 times, Oran: 677 times and Algiers: 1003 times].
- The two algorithms made the same search time [Sidi Bel Abbes, both algorithms are equal: 1272 times, Oran 1043 and Algiers: 738 times].
- On the contrary, the Dijkstra algorithm is faster than the MIGW algorithm in the three cities [Sidi Bel Abbes, Dijkstra is faster 330 times, Oran: 280 times and Algiers: 259 times].

Finally, we can conclude that in the criterion of speed the MIGW algorithm is advantaged compared to its competitor.

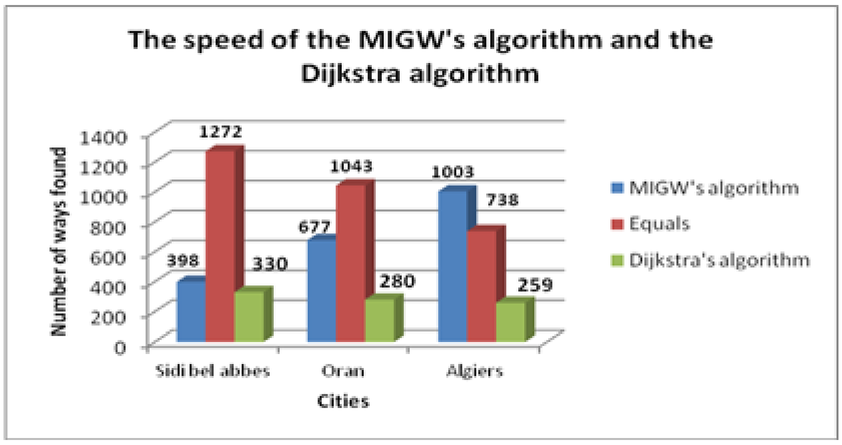

Figure 5: Comparative study between the speed of the Dijkstra algorithm and the MIGW algorithm.

The tests of optimality between the two algorithms, the results confirmed that the two competitors are optimal.

In the three cities of experimentation, the two algorithms succeeded in finding the shortest paths of the requested vertices (Sidi Bel Abbes, success rate = 100.00%, Oran = 100% and Algiers =100%) (Fig: 6).

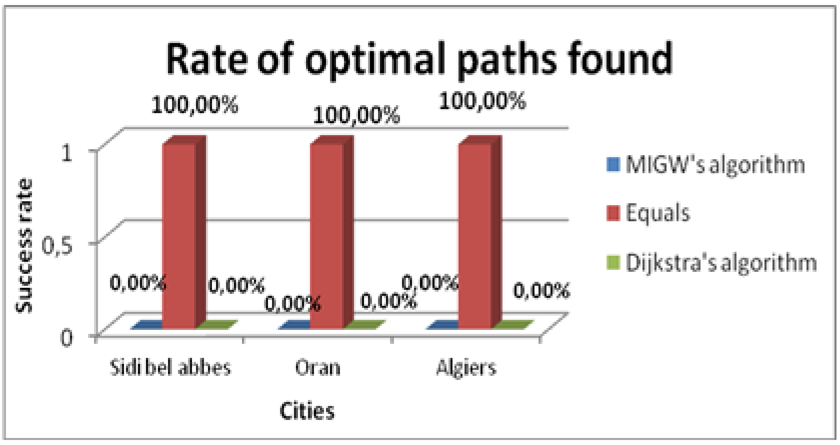

Figure 6: The success rate of the two algorithms (Dijkstra and MIGW) in shortest path search.

The MIGW algorithm has proven from the previous experiments that it is a good competitor to the shortest path search algorithms, because it guarantees optimality and speed.

6. Conclusion:

Path optimization is based on the pattern of the urban road network, planning the best process of routes. This is the heart of the shortest path search. In order to validate the theoretical model of the MIGW algorithm and to propose a decision support system, we built a database of an urban road network of three different Algerian cities [Algiers, Oran, Sidi Bel Abbes]. Experiments have shown that the speed of the MIGW algorithm is influenced only by the size of the solution. Nevertheless, the size of the graph is an important factor in the complexity of shortest path algorithms [e.g. Dijkstra].

Taking advantage of the “multithreading” technique, the MIGW algorithm has given not only optimal results, but very fast results that make it a good competitor to other shortest path algorithms.

MIGW offers us:

- Multiple routes in the same search between the source and destination and the opposite direction.
- The speed of this algorithm depends only on the length of the solution, regardless of the graph of the city.
- With the help of parallelism, the MIGW algorithm has become faster and can be used to solve other more complex problems.

Finally, the strong point of this method is that it offers us several routes in both directions in a short time possible; this variety of solutions is a good platform to solve other problems of optimization of the transport network.

ACKNOWLEDGMENT

This work was partially supported by the MESRS (DGRSDT, PRFU: C00L03UN220120200004).

REFERENCES

[1] H. Sullivan, T. R. Bashkow, and D. Klappholz, "A large scale, homogenous, fully distributed parallel machine, II," in ACM SIGARCH computer architecture news, 1977, pp. 118-124.

[2] T. Hain, "An IPv6 provider-independent global unicast address format," Internet Draft, 2002.

[3] L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP)--Version 1 functional specification," Resource, 1997.

[4] S. Floyd and E. Kohler, "RFC 4341-Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," Internet Engineering Task Force, Request For Comments, 2006.

[5] P. J. Schwartz. (2008). Path search by the A* algorithm, Available: www.developper.com

[6] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in International Symposium on Experimental Algorithms, 2011, pp. 230-241.

[7] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," IEEE transactions on evolutionary computation, vol. 6, pp. 566-579, 2002.

[8] A. Dey, R. Pradhan, A. Pal, and T. Pal, "A genetic algorithm for solving fuzzy shortest path problems with interval type-2 fuzzy arc lengths," Malaysian Journal of Computer Science, vol. 31, pp. 255-270, 2018.

[9] Y. Tsujimura and M. Gen, "Entropy-based genetic algorithm for solving TSP," in 1998 Second International Conference. Knowledge-Based Intelligent Electronic Systems. Proceedings KES'98 (Cat. No. 98EX111), 1998, pp. 285-290.

[10] Y. Nagata and D. Soler, "A new genetic algorithm for the asymmetric traveling salesman problem," Expert Systems with Applications, vol. 39, pp. 8947-8953, 2012.

[11] N. Shanmugasundaram, K. Sushita, S. P. Kumar, and E. Ganesh, "Genetic algorithm-based road network design for optimising the vehicle travel distance," International Journal of Vehicle Information and Communication Systems, vol. 4, pp. 344-354, 2019.

[12] L. Lin, C. Wu, and L. Ma, "A genetic algorithm for the fuzzy shortest path problem in a fuzzy network," Complex & Intelligent Systems, vol. 7, pp. 225-234, 2021.

[13] M. Głąbowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Shortest path problem solving based on ant colony optimization metaheuristic," Image Processing & Communications, vol. 17, pp. 7-17, 2012.

[14] S.-H. Ok, W.-J. Seo, J.-H. Ahn, S. Kang, and B. Moon, "An ant colony optimization approach for the preference-based shortest path search," Journal of the Chinese Institute of Engineers, vol. 34, pp. 181-196, 2011.

[15] C. Mouilah and K. Belkadi, " Application of the "AntNet" in an urban road network: Case of the Algerian communes," in International Conference on Education and e-Learning Innovations, 2012, pp. 1-7.

[16] G. Katona, B. Lénárt, and J. Juhász, "Parallel ant colony algorithm for shortest path problem," Periodica Polytechnica Civil Engineering, vol. 63, pp. 243-254, 2019.

[17] D. Di Caprio, A. Ebrahimnejad, H. Alrezaamiri, and F. J. Santos-Arteaga, "A novel ant colony algorithm for solving shortest path problems with fuzzy arc weights," Alexandria Engineering Journal, vol. 61, pp. 3403-3415, 2022.

[18] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," Information processing letters, vol. 103, pp. 169-176, 2007.

[19] Y.-F. Liao, D.-H. Yau, and C.-L. Chen, "Evolutionary algorithm to traveling salesman problems," Computers & Mathematics with Applications, vol. 64, pp. 788-797, 2012.

[20] Y. Deng and H. Tong, "Dynamic shortest path algorithm in stochastic traffic networks using PSO based on fluid neural network," Journal of Intelligent Learning Systems and Applications, vol. 3, p. 11, 2011.

[21] C. Dudeja, "Fuzzy-based modified particle swarm optimization algorithm for shortest path problems," Soft Computing, vol. 23, pp. 8321-8331, 2019.

[22] S. Alani, A. Baseel, M. M. Hamdi, and S. A. Rashid, "A hybrid technique for single-source shortest path-based on A* algorithm and ant colony optimization," IAES International Journal of Artificial Intelligence, vol. 9, p. 356, 2020.

[23] H. Arslan and M. Manguoğlu, "A hybrid single-source shortest path algorithm," Turkish Journal of Electrical Engineering & Computer Sciences, vol. 27, pp. 2636-2647, 2019.

[24] M. R. S. Aghaei, Z. A. Zukarnain, and A. Mamat, "A hybrid algorithm for finding shortest path in network routing," 2009.

[25] S. Zhang and Y. Zhang, "A hybrid genetic and ant colony algorithm for finding the shortest path in dynamic traffic networks," Automatic control and computer Sciences, vol. 52, pp. 67-76, 2018.

[26] C. Mouilah and A. Rahmoun, "A Balanced Traffic Routing Using the Bio-inspired Traversing and Marking Metaheuristics," Rev. d'Intelligence Artif., vol. 34, pp. 39-44, 2020.