Combining VNS with Genetic Algorithm to Solve the One-to-One Routing Issue in Road Networks

Omar DIB[a,b], Marie-Ange MANIER[b], Laurent MOALIC[b], Alexandre CAMINADA[b]

[a]*Technological Research Institute SystemX, 91120 Palaiseau, France*
omar.dib@irt-systemx.fr
[b] *OPERA-UTBM , 90010 Belfort, France*
{omar.dib, marie-ange.manier, laurent.moalic, alexandre.caminada}@utbm.fr

Abstract

Computing the optimal route to go from one place to another is a highly important issue in road networks. The problem consists of finding the path that minimizes a metric such as distance, time, cost etc. to go from one node to another in a directed or undirected graph. Although standard algorithms such as Dijkstra are capable of computing shortest paths in polynomial times, they become very slow when the network becomes very large. Furthermore, traditional methods are incapable of meeting additional constraints that may arise during routing in transportation systems such as computing multi-objective routes, routing in stochastic networks. Therefore, we have thought about using meta-heuristics to solve the routing issue in road networks. Meta-heuristics are capable of copying with additional constraints and providing optimal or near optimal routes within reasonable computational times even in large-scale networks. The proposed approach is based on a hybridization process done between a genetic algorithm (GA) that belongs to the population-based metaheuristics and a variable neighborhood search (VNS) that performs with one single solution. To evaluate our method, we made experimentations using random generated and real road network instances. We compare our approach with two exact algorithms (Dijkstra and integer programming) as well as with GA and VNS when they are executed solely. Experimental results have proven the efficiency of our approach in comparison with the other approaches.

Keywords: Routing; Road networks, Shortest Path Problem, Exact approaches; Dijkstra; Integer Programming; Meta-heuristics, Genetic Algorithm; VNS; Hybrid meta-heuristics

1. Introduction

Finding the best route to reach a destination in road networks is not always straightforward. Travelers usually encounter difficulties when planning their trips in elaborate road schemes. Having several possibilities to go from one place to another usually makes travelers incapable of figuring the best route out. Consequently, to help travelers route planning has gained significant importance in recent years. Several commercial navigation products have been developed to provide travelers with driving directions helping them to reach their destinations.

Finding the optimal path between two places in road networks refers to solving the one-to-one shortest path problem (SPP) in a directed connected graph. Graph's nodes represent road junctions and edges connecting two nodes account for road segments. The shortest path problem (SPP) is one of the best known problems in combinatorial optimization. It is widely applied in different fields such as transportation, telecommunication and computer networks. It aims at determining the path between two points with the minimum associated/allocated resource (distance, time, cost…) in a directed, undirected or mixed graph. Although, the SPP has been well studied over the last 60 years, researchers still endeavor to extract new variants of the basic problem in order to model and provide efficient solutions for several real world issues.

Important forms of the SPP may be considered such as finding the shortest paths in a dynamic graph where the arcs' weights vary as functions of time. Moreover, finding shortest paths subject to some additional constraints like

fixing the maximum number of visited nodes, minimizing the largest edge cost (bottleneck SPP) or the difference between the largest and smallest edge cost (balanced SPP) etc.To deal with the SPP and its different variants, several algorithms have been developed since 1956. The standard solution to the one-to-one shortest path problem, which is the target of our study, is Dijkstra's algorithm [1]. For the average case, one can get an O ((|E |+|V|) * log |V|) (linear : |V| #Vertices, |E| #Edges). To deal with negative edge weight, we have the Bellman-Ford algorithm [2]. Floyd-Warshall algorithm [3] was also proposed to find shortest paths between all pairs of points in a weighted graph with positive or negative edge weights. Its running time is O ($|V|^3$) in the worst case. For sufficiently dense graphs, it is usually faster than |V| calls to Dijkstra's algorithm.

The SPP was also formulated and solved using Integer Programming (IP) techniques but experimental results showed that its running time is usually higher than standard algorithms. IP is widely used for solving many variants of SPP when standard algorithms fail due to the complex nature of additional constraints, but this usually leads to a significant running time. Computing best routes in road networks can be done using Dijkstra's algorithm or even the IP techniques. However, this would be far too slow for users seeking answers in milliseconds, especially in large-scale networks. For instance, Dijkstra would take up to 10 seconds to solve a shortest path query in a continental road network. Therefore, extensive research have been done in order to accelerate traditional algorithms.

For example, the running time of Dijkstra can be enhanced using bidirectional search. Other techniques have been developed to accelerate routing in road networks such as A* search [4], Arc Flags and ALT (A*, landmarks, and triangle inequality) [5], etc. Although, those techniques are fast enough to provide driving directions even in large-scale road networks, they become less performant or even inapplicable when additional constraints are added to the SPP such as dynamic arcs weights, multimodal shortest paths and multi-criteria paths optimization. To overcome such shortcomings, we believe that meta-heuristics such as Genetic Algorithms, local search procedures are efficient candidates to find optimal or near optimal solutions within an acceptable computation time.

Meta-heuristics such as such as Genetic Algorithms (GAs) [6], Particle Swarm Optimization (PSO) [7], Ant Colony Optimization (ACO) [8] have been applied for solving routing issues in various fields of applications. For instance, reference [9] proposed a genetic algorithm to find the shortest path in computer networks. Moreover, reference [10] used a genetic algorithm to solve the routing issue in road networks. Reference [11] also worked with genetic algorithm to find the shortest path in data networks. Reference [12] combined PSO with local search and velocity re-initialization for computing shortest paths in computer networks. Although the previous mentioned works are very interesting, we have remarked that experimental results have been only done over small network instances (ex: 100 nodes). Therefore, there is no guarantee that such approaches continue to be performant on large-scale networks. Furthermore, none of the such methods has been applied to solve other complex variants of SPP such as computing multi-objective shortest paths in stochastic networks.

In this paper, we propose a new approach for solving the routing issue in road networks. We introduce a new combination process in which we couple variable neighborhood search with genetic algorithm. The remainder of this paper is organized as follows. In next sections, we present more formally the one-one-one SPP. We discuss in section 3 some theoretical features of two exact methods [13] used to solve the SPP Dijkstra and IP technique. We move on then to talk about meta-heuristics [14] and precisely Genetic Algorithm (GA) [15], variable neighborhood search (VNS) [16] and the hybrid approaches. Section 5 is devoted to explain our hybrid approach. We present and discuss in sections 6 and 7 the experimental results. Lastly, section 8 is devoted to conclude this paper as well as to propose some future works.

2. Shortest Path Problem

Behind all routing or route planning problems is the concept of the shortest path. This latter is one of the most classic problems in graph theory, and has been investigated extensively since half a century. The one-to-one SPP addressed in this study is defined as that of finding the path with the minimum cost (time, distance...) path between a given source and destination. More formally, given a directed graph $G = (V, E)$ with node set V of cardinality n, edge set E of cardinality m and costs $c(e) \in R$ for all edges $e \in E$. The one-to-one version of SPP is then asks for finding a path p ($e1, e2... ek$) between a given source node *'s'* and a destination *'t'* in such a way the sum of the edges' weights in p is minimized.

3. Metaheuristics

Meta-heuristics can be defined as search algorithms, which were initially designed to find satisfying approximated optimal solutions in a reasonable computation time for several optimization problems. In contrast to conventional algorithms, Meta-heuristics do not iteratively visit each element in the search space. They actually guide the search process towards regions that may contain high quality solutions. Meta-heuristics can be classified under two main categories [19]: single solution and population based search. The former proceeds with one initial solution and tries to enhance its quality by repeatedly moving to a better solution. Examples of such type of algorithms are simulated annealing (SA) [20] and tabou search (TS) [21]. However, population-based are meta-heuristics in which a population of solutions is maintained at each iteration. Evolutionary algorithms [22] are good examples for population-based algorithms.

3.1. Genetic Algorithms (GAs)

Genetic Algorithms (GAs) are known as adaptive heuristic search algorithms. GAs are inspired by biological evolution and they are based on the idea of the survival of the fittest. The power of GAs lies in their ability to avoid randomness search by intelligently exploit historical information to direct the search process towards the regions of better performance within the solution space. GAs ultimately converges to the individual which is the best adapted to the environment and thus obtaining the optimal or a satisfactory solution. After generating a set of initial solutions, GAs evolve through three main genetic operations: **Selection**, **Crossover**, and **Mutation**.

- **Selection:** In this phase, the algorithm decides which individual (solution) will survive or pass its genes to the next generation. Better solutions have always more chances to survive. The selection operation is therefore to prevent the loss of effective gene to make high performance individuals survival with greater probability, thereby enhancing the global convergence and computational efficiency.
- **Crossover:** The crossover operation is used to assemble a new individual, and do effective search in the solution space. There are several techniques to accomplish the crossover such as single point, multiple points, and variable point crossovers. The efficiency of those latters depends mostly on the problem.
- **Mutation:** This operator is usually used to maintain diversity and to prevent premature convergence. It is often performed with low probability since it results in a random walk through the search space.

3.2. Variable Neighborhood Search (VNS)

VNS is a single solution meta-heuristic that is firstly proposed by Hansen and Meladenovic in 1997. VNS has been proved to be very useful for obtaining high quality approximate solutions to several optimization problems. VNS can be used for solving wide range of optimization problems ((linear, nonlinear, continuous, discrete, integer program problems, mixed integer program problems etc. . .). The high performance of VNS is stemmed from its ability to explore distant neighborhoods of the current incumbent solution [22]. VNS can move from a solution to another by repeatedly applying local search improvements to reach local optima. Once a local optimum is detected, VNS is able to jump out of it and eventually find better solutions by dynamically changing the neighborhood‘s structures. Therefore, VNS is more likely to prevent the optimization process from rapidly falling into local optima. VNS has several advantageous in comparison with conventional meta-heuristics such as tabu search, simulated annealing. The algorithm is not only easy to achieve since its structure is simple, but it is also irrelevant to the problem and is suitable for all kinds of optimization problems. Several extensions have been derived from VNS such as Variable Neighborhood Descent (VND), Reduced VNS (RVNS) [22], skewed VNS (SVNS) [23], etc... These variants have been invented in order to enhance the performance of the basic VNS scheme as well as to adapt it for different optimization scenarios.

3.3. Hybrid Metaheuristics

Hybrid meta-heuristics have received considerable interest in combinatorial optimization in recent years. The main motivation for the hybridization process is to obtain better performance by exploiting and combining advantages of the individual pure strategies [24]. We introduce in this paper a new hybridization approach in which we use the local search procedure VNS inside the population-based meta-heuristic GA.

The hybridization process done between single solution and population-based meta-heuristics refers nowadays to the term memetic algorithm [25]. The term (MA) is inspired by both Darwinian principles of natural evolution and Dawkins' notion of a meme. It was first introduced in 1989. MAs were first invented to reflect the fact that coupling genetic algorithms (GAs) with individual learning procedure may perform local refinements. Nowadays, MAs refer to a hybridization process done between population-based meta-heuristics such as GAs and single-point search or local search procedures such as simulated annealing (SA) [19] and tabu search (TS) [20]. MAs have proved to be effective in solving a wide range of optimization problems such as graph coloring, scheduling etc.

4. Proposed Approach : Hybrid meta-heuristic

As aforementioned, the main contribution of this paper is to adapt and apply an approximate method, which is based on a collaboration between two meta-heuristics, for solving the one to one SPP. The meta-heuristics used are GA that belongs to the population-based algorithms from one side and VNS that belongs to the single point search algorithm from the other side. Our method proceeds with a population of solutions as GAs works. However, in our case, initial solutions are generated using a double search algorithm that is able to provide feasible paths between any two nodes. The details of this algorithm is described later in this article.

Once a population of solutions (feasible paths) are successfully generated, we did an enhancement operation over each individual in the first population. That is, we apply a VNS over the first population. We decided with improving initial solutions since their quality would possibly help the algorithm find the optimal solution. More details about VNS adaptation to the problem will be discussed later. Once we accomplish the VNS, we send individuals to an evaluation process. Our fitness method is the sum of the weights of each edge including in a path.

After the improving phase, we repeatedly perform certain number of operations in the goal of increasing the algorithm’s chance to find the optimal or high-quality paths between the origin point and the destination one. We begin with the crossover operation that is responsible for forming two new paths (offspring) from two initial solutions. Crossovers are usually used to exploit new range of the search space. We have used single-point as a crossover technique. We will discuss later and in more details the crossover operation.

After that phase, a new population of solutions is created. Thus, new paths between the origin and the destination point have been probably detected. Therefore, re-applying the VNS procedure to the new population will possibly enhance the paths’ qualities. That is, we perform a special mutation operation based on VNS method over each individual (path) in the population. Thanks to this technique, our algorithm will have more chances to exploit and explore new regions of the search space. The whole process is repeated until the algorithm reaches our stopping criteria. The following scheme represents the algorithms’ steps that we will discuss in more details in next paragraphs.

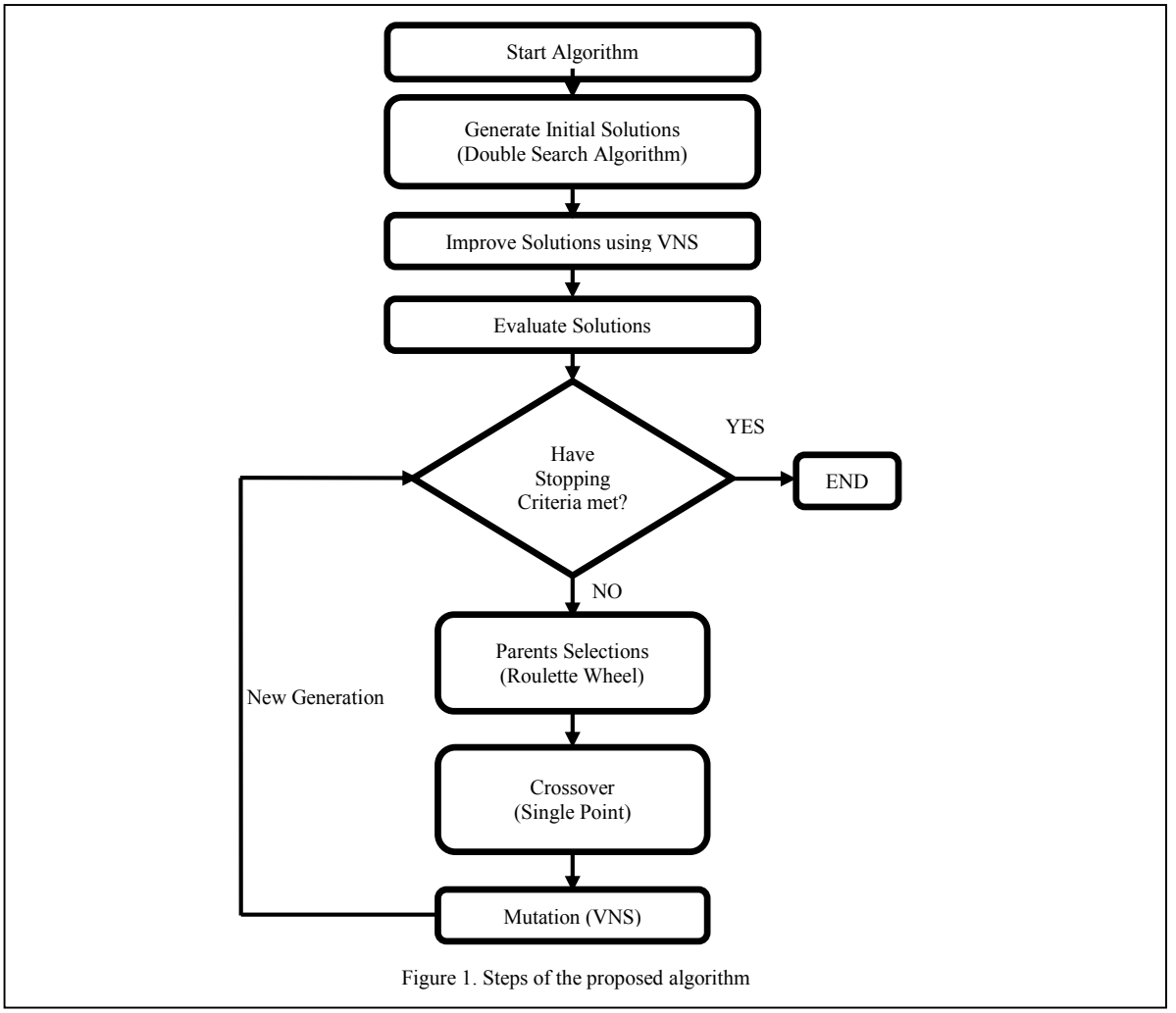


Figure 1. Steps of the proposed algorithm

4.1. Encoding

Encoding of individuals is a crucial step in GA. It is largely dependent on the property of the problem, and will affect the design of genetic operation. We have used in our work the permutation encoding to represent a chromosome in a population. Each individual is encoded as a string of numbers, which represents the identifiers of edges in a path. The size of each chromosome is not equal. For example, in the following graph, the path to get from A to E is encoded as a vector P = (AB, BC, CD, DE) in which each element is the identifier of the correspondent edge. Since several paths with different nodes and edges may exist to go from an origin point to a destination one, the dimension of chromosome is not fixed.

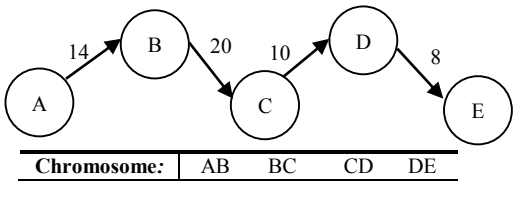


Figure 2. Encoding of chromosomes

4.2. Generating Initial Solution

Generating initial solutions for meta-heuristics is not always straightforward. Usually, good initial solutions might rapidly guide the search process towards important regions in the search space. Our initial solutions are a set of feasible paths generated using a double search algorithm. We have modified a double search process that simultaneously run a forward search from the origin point (s) and a backward search from the destination point (t) in order to get a set of feasible paths between a pair of nodes.

The algorithm works as follows:

- It initially defines two queues: one for the forward search "FQ" and another for the backward "BQ". The source and target nodes "s" and "t" are then added to "FQ" and "BQ" respectively.
- After initialization, the algorithms starts repeatedly taking the element at the head of "FQ"; adding its adjacent nodes to "FQ"; removing it from "FQ". The same process is also done from the backward search.
- To keep a history of the search processes, nodes visited from the forward search are assigned an "f" flag and nodes visited from the backward search are assigned a "b" flag. Moreover, we store the incoming edges that allow the algorithm to reach each forward node and the adjacent edge of each backward node.
- Once the algorithm is about to add in "FQ" a node that has already been visited from the backward search, it means that a path between 's' and 't' has been detected. Therefore, the procedure of constructing that path starts. The same process is also performed from the backward search.
- To reconstruct a path found, we have used the historical information stored in each node object.
- The algorithm terminates if one of the queues "FQ" and "BQ" becomes empty. We have remarked also that in some graph instances our algorithm may provide so many initial paths. We have added therefore the maximal number of generated paths as another termination criterion for this case. By doing so, we can control the size of the first population.

Example: Here is an example to explain more the algorithm. Some simplifications have been made to reach better understanding. Given a graph G (Figure 4) with 6 vertices and 10 edges with positive weights. To get initial feasible paths between the origin node 'S' and the destination point 'T'. Our algorithm performs as follows.

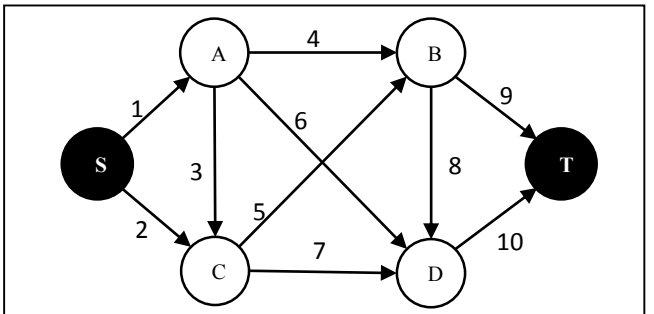


Figure 3. Generating initial feasible paths between 'S' and 'T'

Step1: It initializes a forward queue FQ and a backward queue BQ

Step2: It adds 'S' to FQ and 'T' to BQ

FQ=S	*BQ=T*

Step3: It initializes a table to maintain a history about our search. In another term, we precise from which search type each node has been visited. For simplicity, we make 'f' for "forward" and 'b' for backward. Moreover, we store the edge that leads to reach each visited node.

Table 1. History table for each node

Node	*S*	*A*	*B*	*C*	*D*	*T*
Flag	f	-	-	-	-	b
Edge	-	-	-	-	-	-

Step4: It adds the adjacent nodes of the head element of FQ and remove that element from FQ.

FQ=	~~*S*~~	*A*	*C*

Step5: It updates the search history table

Table 2. Updating the history of each visited node

Node	*S*	*A*	*B*	*C*	*D*	*T*
Flag	f	f		f	-	b
Edge	-	(SA)	-	(SC)	-	-

Step6: It adds the incoming nodes of T to BQ and remove T from the queue.

BQ=	~~*T*~~	*B*	*D*

Step7: It updates the search story

Table 3. Updating the history table

Node	*S*	*A*	*B*	*C*	*D*	*T*
Flag	f	f	b	f	b	b
Edge	-	(SA)	(BT)	(SC)	(DT)	-

S*tep8:* It takes the node A form FQ and try to add its adjacent nodes. The first adjacent node is C. However, C has been already visited from a forward search. Thus, nothing will change. The algorithm tries then to add B to the queue, but B has already the 'b' flag. Therefore, the algorithm realizes that a path between 'S' and 'T' is detected.

Step9: the algorithm constitutes the path found. To do so, it takes the edge where the two searches intersect. In this case, it is the edge (AB). The algorithm then uses the search history table to build the path from the node 'A' to 'S' and from the node 'B' to 'T'.

Path1:	SA	AB	BT

The algorithm will continue searching for other paths while the queues are not empty. After continuing the process, the algorithm will find three different paths in addition to path1.

Path2:	SA	AD	DT

Path3:	SA	AC	CB	BT

Path4:	SC	CD	DT

Finally, as the algorithm performs, there is a chance that two identical paths are found between 'S' and 'T'. To tackle such problem, the algorithm keeps a history about the intersection edges found during the execution. It then ignores a path if the intersection edge has been already added to the list.

4.3. Enhancing Initial Solutions Using VNS

Since the quality of initial solutions may affect the performance of meta-heuristics, we decide with enhancing our initial solutions. To do so, we have used a VNS method as an enhancement operator. Indeed, we have applied a VNS method over half of the individuals in the first population. This technique will increase the diversity level of the initial population as well as establish a good repartition of solutions within the search space. Consequently, the algorithm's chance to find the optimal solution will increase.

In the following, we will explain how the VNS method has been adapted to deal with our problem. Indeed, one of the major challenges addressed in VNS is the construction of the neighborhood structures. To deal with that issue, our algorithm performs a preprocessing operation during the generation phase of the network.

The idea is to examine each edge in the graph and check if its starting and ending nodes have a common node. Two nodes A and B have a common node if and only if the end point of one adjacent edge of A is the same as the starting point of an incoming edge of B. After doing that, we will end up with two neighboring structures that we will use when applying VNS.

In the graph below, if we take the edge (AC), we can notice that the node B is shared between the adjacent edge (AB) of A and the incoming edge (BC) of the node C. Hence, we can deduce that we have an alternative path to reach the node C from A, which is in this case the path (AB, BC).

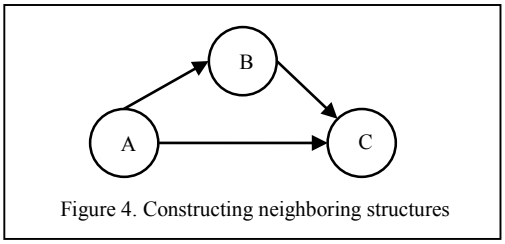


Figure 4. Constructing neighboring structures

Once we find such common nodes, we can start constructing our neighboring structures as follows. If the length of the edge (AC) is greater than the length of the path (AB, BC), thus, whenever we meet the edge (AC), we can replace it by the path (AB, BC). That case makes our first neighboring structure. That is, our first neighboring structure is a list containing replacement paths formed by two edges for each edge. A second scenario may arise if the length of the path (AB, BC) is greater than the length of (AC), thus, we can replace the path (AB, BC) by the edge (AC) in any path. Our second neighboring structure is then a list containing an edge that will replace a path formed by two edges.

Figure 6 shows a graph with 13 edges and 8 nodes. An example of the replacements included in the first neighboring structure is to substitute the edge (SC) with the 2-edges path (SA, AC). Replacing the 2-edges path (BD, DT) by the edge (B, T) is an example of an instance existed in the second replacement structure.

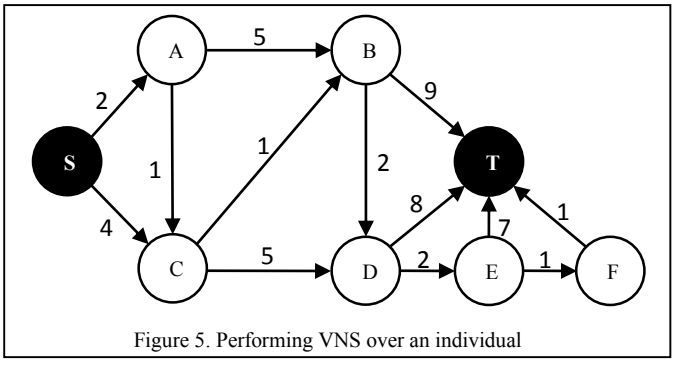


Figure 5. Performing VNS over an individual

After performing the preprocessing operation over the graph above, we end up with two replacement lists that illustrate our neighboring structures.

Table 4. Items in the first replacement list

Edge	***Replacements***
(SC)	(SA, AC)
(AB)	(AC, CB)
(CD)	(CB, BD)
(ET)	(EF, FT)
(DT)	(DE, EF, FT)

Table 5. Items from the second replacement list

Path	***Replacements***
(BD, DT)	(BT)
(DE, ET)	(DT)

After carefully examining the two replacement lists, we have realized that the performance of our VNS can be improved by applying some enhancement operations. More specifically, we try to enhance the quality of solutions that we can get by applying the first structure using information from the two replacements list.

For instance, the path (DE, ET) included in the second list can be replaced by the edge (DT). However, if we carefully scrutinize, we can notice that the edge (ET) is included in the first structure and the length of its replacement path added to the length of (DE) is shorter than the length of (DT). Therefore, we can add a new element to the first structure and it will contain the substitution of (DT) by the path (DE, EF, FT).

We can also improve the quality of elements in the first list by using the list itself. For instance, let us imagine that the edge (ET) has a weight of 5. The first list will then contain a row containing the replacement of (DT) by (DE, ET). However, (ET) is also in the list. Hence, we can replace (DT) by (DE, EF, FT).After the enhancement operations, our algorithm gets the chance to find better solutions and thereby its performance will increase. By taking the path *P*= (SC, CD, DE, ET) which is a solution to go from *S* to *T*,We can say that the neighbor solutions of P that we get by using the first neighborhood structure are:

P_{11}:	SA	AC	CD	DE	ET
P_{12}:	SC	CB	BD	DE	ET
P_{13}:	SC	CD	DE	EF	FT

The neighbor solutions of *P* after applying the second neighborhood structure are:

P_{21}:	SC	CD	DT

After defining the two neighboring structures, we want now explain the mechanism that we adopted to move from one solution to another. As shown in Figure 7, the VNS method takes as input a set of neighboring structures and a single initial solution and it performs then some operations in order to enhance its initial solution.

Our VNS proceeds with the first neighboring structure. It examines all the neighbors' solutions belonging to the current solution. VNS then selects the best neighbor among all neighbors according to the fitness. More the fitness is higher, more the solution is better. VNS then compares the best-selected neighbor with the current solution; it moves to the best neighbor if its fitness is better than the current solution. If not, VNS changes the neighboring structure. The process is repeated while the current solution can be enhanced by using any of the neighboring structures.

Input:

- Two neighboring structures N_K, (K=1,2);
- An individual (solution) x;

Iteration

```
1.  While stopping rule is not satisfied do
2.      k=1;
3.      While k<2 do
4.          Exploration of neighborhood: find the best neighbor x' of x (x'€ Nk (x))
5.          Move or Not:
6.          If f(x') < f(x) then
7.              x ← x', k ← 1;
8.          else
9.              k ← k+1;
10.         end if
11.     end while
12. end while
13. return the best solution
```

Figure 6. Variable Neighborhood Search

Assuming that the path x = (SC, CB, BD, DT) in Figure 6 is an initial solution to the VNS method with length 15. To apply VNS, the algorithm performs as follows:

It uses firstly the neighboring structure; It then calculates all the neighbors solutions; it then selects the best neighbor. Neighbor solutions of x using the first neighboring structure are :

x_{11}:	SA	AC	CB	BD	DT	Fitness=14

x_{12}:	SC	CB	BD	DE	EF	FT	Fitness=11

Neighbor solutions of x using the second neighboring structure are:

x_{21}:	SC	CB	BT	Fitness=14

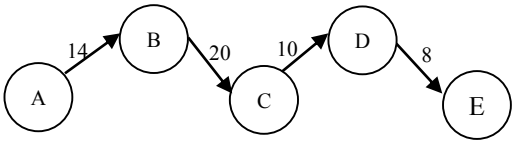
The best neighbor of x using the first neighborhood structure is x_{12}. The algorithm will then move to x_{12} since its fitness is better than x. The same process is repeated over x_{12}. The best neighbor of x12 using the first neighborhood structure is the path x*= (SA, AC, CB, BD, DE, EF, FT) with length 10. The algorithm moves to x* since its fitness is strictly better than x_{12}.

The same process is repeated over x*. However, x* cannot be enhanced either by using the first neighboring structure or the second one. Therefore, the algorithm stops and return x*. Finally, our VNS method is based on two-neighboring structures. At each time our algorithm gets stuck at a local minimum, we change the structure of the neighborhood. By following this technique, our algorithm will exploit and explore wide regions of the search space. Adding more than two neighboring structures will probably enhance the chance of the algorithm to find better solutions. However, that might increase the time to accomplish the preprocessing operation as well as the time to perform the VNS itself.

The strategy we have adopted to move from one solution to another is the best enhancing neighbor. Other techniques can be used such as “The first enhancing neighbor” or the “least enhancing neighbor”. Such techniques may provide better results and extensive research are conducting nowadays to decide which technique is the most performant. Unfortunately, we haven’t have the time to implement and compare the results of the different moving techniques. However, future works have been planned to be done in order to study the impact of the moving technique on the performance of the VNS method.In addition, we have randomly chosen to apply the first structure neighborhood before the second one. In fact, deciding which neighboring structure should be applied at which order has been always an issue for VNS method. Further works will be done in near future to study if the order of neighboring structure will affect the performance of our approach.

4.4. Evaluating an Individual

The evaluation function for our method is nothing but taking a path in the population and adding the weights between each node pairs. The result is a non-negative number that represents the path length. The cost to get from one point to another is stored as a parameter into the edge entity.



Chromosome:	**AB**	**BC**	**CD**	**DE**
Fitness:	14+20+10+8=52			

Figure 7. Fitness evaluation

4.5. Crossover

After evaluating individuals, we have repeatedly applied, with different probabilities, crossover and mutation operations in order to enhance current solutions.To accomplish the crossover process, a single tournament selection mechanism is employed. After ordering the individuals in the population, each chromosome in the odd position is mated with the next chromosome in order to produce new individuals (offsprings). By doing so, we produce a new population having twice the size of the current population. The best half individuals are then selected for the next generation and the rest are ignored. However, there is a big chance that the same individual is duplicated in the population as the generations go on. We therefore, replace the duplicated individuals with newly generated chromosomes. This process will undoubtedly increase the diversity within the population.

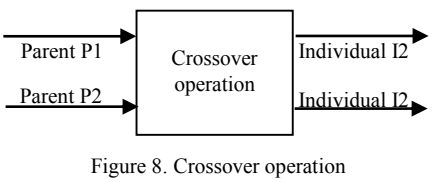


Figure 8. Crossover operation

Single point crossover technique has been used in our approach in order to produce offspring. An intersection node between two individuals is selected to be the crossover point. Current individuals exchange then part of genes with each other before or after the crossover point to generate offsprings.

Assuming that G is a directed graph with eight edges and seven nodes. Let us also assume that from an initial generated population, two paths are found between A and T.

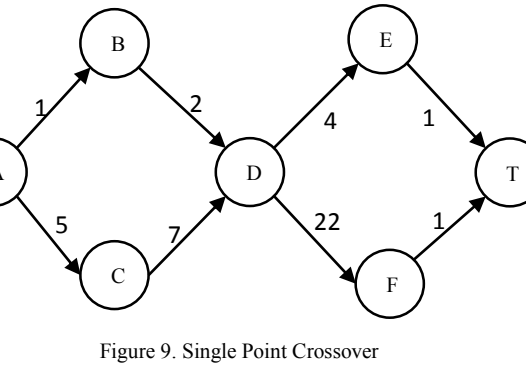


Figure 9. Single Point Crossover

The first path is the sequence of the following edges (AB, BD, DE, ET); its length is 13. The second path is (AC, CD, DF, FT) with length 17. As can be noticed from those sequences, they have in common the node D. Over that point, we will perform the crossover. That is, we construct two new paths (offsprings) from two initial individuals. Each offspring is made up of two sequences that belong to different paths

Table 6. Performing Single Point Crossover

P1:F=13	AB	B***D***	***D***E	ET
P2:F=17	AC	C***D***	***D***F	FT
C1:F=8	AB	BD	DF	FT
C2:F=22	AC	CD	DE	ET

P: Parent **C**: Child **F**: Fitness

The offsprings produced after applying our operator are C1= (AB, BD, DF, FG) with length 8 and C2= (AC, CD, DE, EG) with length 22. We can easily remark that our algorithm gets the chance to explore a new path with less cost, which is 8 and it represents the optimal path for our example. We can easily remark that our algorithm gets the chance to explore a new path with less cost, which is 8 and it represents the optimal path in our example. One point worth mentioning is that after the crossover operations that we used in our method, we do not care about the feasibility of the new path generated. We will always end up with a feasible path from the source to the destination. Therefore, the algorithms does not lose time to test the validity of the path nor to perform some additional operations to repair the wrong path.Finally, experimental results show that using more than one crossover point might increase the diversity of our approach. Therefore, the algorithm's chance to find better solutions will also increase. However, that may be at the expense of additional computational efforts. We therefore decided with only using the simple crossover technique.

4.6. Mutation (VNS)

Crossover operation may produce degenerate population. The algorithm may therefore get trapped at local minima. To overcome this issue, we perform the mutation operation. We have chosen the VNS as a special mutation operator. That is, we have applied VNS over the population's individuals. By doing so, we guide the algorithm towards new regions within our solution space. We therefore increase the algorithm's chance to find better solutions. Furthermore, applying VNS will ensure that the genetic algorithm maintains a sufficient diversity level that prevents premature convergence. The mutation operation has been applied at each time we perform the crossover. The conventional mutation operation used in GAs is usually applied with low probability. However, in our memetic approach, the mutation is always applied after crossover. The purpose of doing that is to allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.We did apply other mutation techniques such as order changing, but we have realized that the mutation becomes less performant and it may provide invalid paths. An additional process should therefore be applied to reform infeasible paths. As a result, the mutation computational time will increase. We decided thereby against using such traditional mutation techniques.

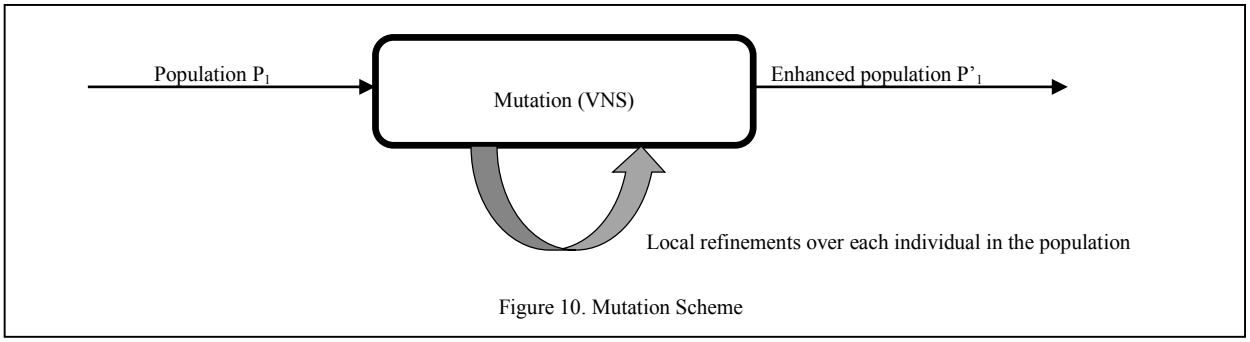


Figure 10. Mutation Scheme

4.7. Terminating Condition

Approximate methods do not guarantee that the final solution found during execution is the optimal one. They do not therefore have the ability to automatically stop performing when the best solution is detected. Additional terminating conditions should then be introduced in order to allow the convergence of the algorithm.

Maximum number of generations, fixed execution time, and no modifications in population elements are considered as algorithm stopping criteria. We have used in this study two stopping criteria. Firstly, the algorithm stops when it fails to find better solutions during several continuous steps. We have used 100 generations as a number to ensure a fixed state in the population. Another stopping criterion is until the algorithms reaches the maximum number (say 500) of generations. We have noticed after some experimentations that our algorithm visits wide range of the search space rapidly. Thus, there is a big chance that the algorithm converges after few generations. That explains the small number of generations used as stopping criteria.

5. Experimental Results

We have done experimentations over 50 different graphs instances (from small to large size). For each instance, we choose five-times two random points to construct our origin-destination queries. We have used instances belonging to three categories in order to test our approach.

- **Random graphs** which are generated thanks to a graph generator that we developed using Java.
- **Complete graph**s where each node is linked to all other nodes
- **DIMACS graphs** which represent real-world road networks benchmarks taken from the DIMACS challenge website that offers graphs for road networks in USA.

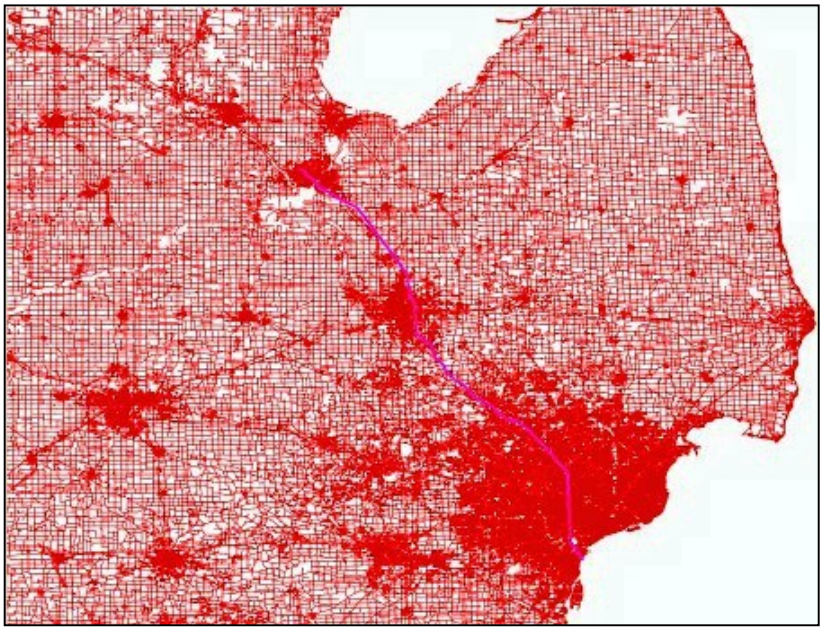


Figure 11. A real road network from DIMACS Challenge with 2,758,119 nodes and 6,885,658 edges

We have compared our hybrid approach with two exact approaches (Dijkstra and IP) and two meta-heuristics GA and VNS executed separately. We implemented Dijkstra using a priority queue and we solved the IP generated using two solvers: CPLEX 12.6 and GUROBI 6.0. The GA executed separately has the traditional GA steps. We generated initial solutions thanks to the double search algorithm we developed for this issue. In contrast to our approach, we do not enhance initial solutions in this case. Simple crossover technique is applied with a high probability of 0.9. Mutation in this case is not performed using VNS. It is however done by using the replacements lists that we developed to perform the VNS. Indeed, we take an individual, and we search for substituting any edge in the path. The mutation is performed with a low probability of 0.1. By using the traditional mutation technique, we only make a small perturbation in the solution. Therefore, the diversification level is very low. The VNS executed separately takes an initial solution generated thanks to the double search algorithm developed earlier. It then tries to enhance its quality untill it reaches the stopping criteria.

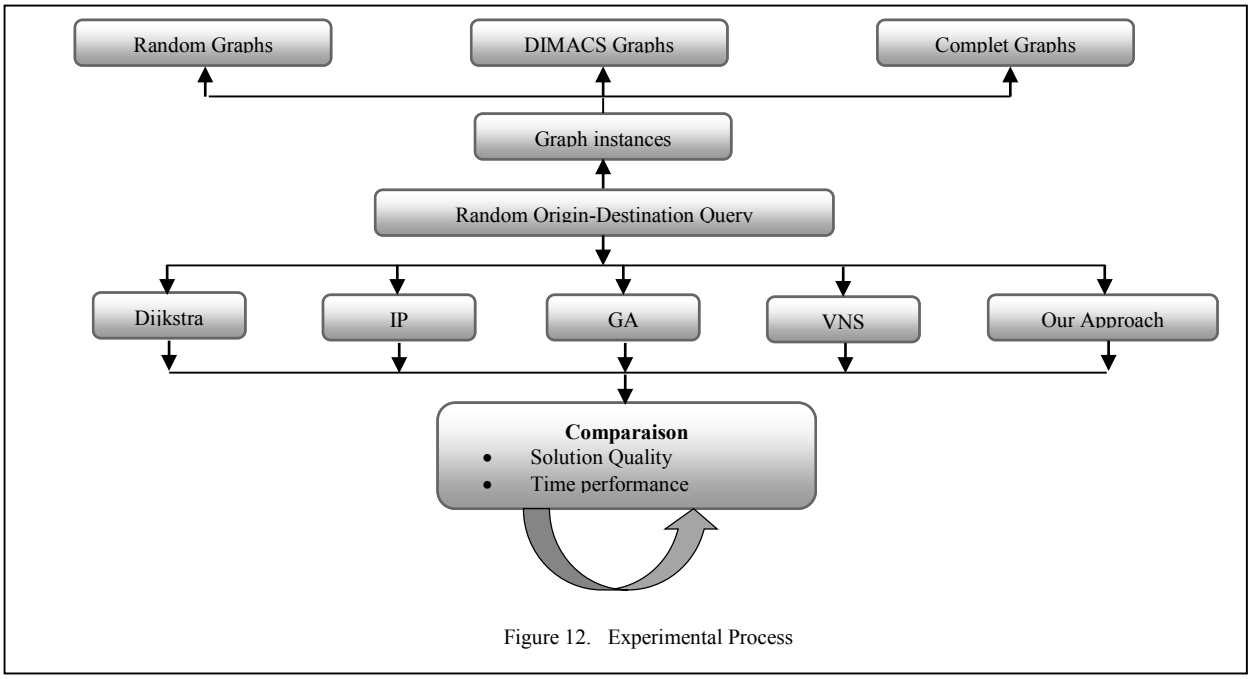


Figure 12. Experimental Process

We run the used algorithms and solvers on an Intel core I5 machine of 8 GB RAM. Moreover, we made extensive use of generic programming techniques in order to avoid runtime overheads. We also put particular efforts into carefully implementing efficient data structures. For instance, to represent a graph, we used our own tools instead of using existing libraries, which usually entail certain undesired overheads.

The comparison between the different implemented approaches is done by evaluating these factors:

- Solutions quality
- Time performance (efficiency)

Measuring the solution quality of an approximate method refers to calculating the gap between the best solution found by the method and the optimal value of the problem.

The gap formula we have used in our study is defined as:

$$\text{Gap} = \frac{v_{opt} - v_{best}}{v_{opt}} * 100$$

Where v_{opt} is the optimal path found by an exact method, and v_{best} is the best path found by our method.

The time performance of a method refers to the CPU time spent to solve the problem. By dividing the CPU time of two method, we can calculate the average speed of a method in comparison with the other. We have compared in our results the running times of the implemented approximate methods with the running time of Dijkstra. The result of comparison gives the average speed of each approach compared with Dijkstra.

The results showed that the running time of Dijkstra is highly better than IP weather it is solved by CPLEX or by GUROBI. However, our method outperforms Dijkstra and IP with 5% average GAP to the optimality. Its average speed is 20-times faster than Dijkstra and more than 1000-times compared to IP.

Results also showed that applying the basic GA solely would always outperform our approach as well as the other methods except VNS. However, the average GAP to the optimality in this case is usually very high (~20%).Furthermore, the time performance of solely applying VNS is always better than all the other approaches. However, its average GAP to the optimality is very high (30%).

Moreover, we have noticed from the results that the time spent to solve an integer program by the CPLEX solver is not the same as in GUROBI solver. It changes at each request. Although this research does not focus on accelerating the IP's solving time, we believe that the techniques used in both solvers can be enhanced. That point may be considered as a future work since it can tell us when and how we should change the parameters of our solvers depending on the problem instance.

Furthermore, regardless the efficiency of our approach, we remarked that in special instances of graphs, it fails to outperform Dijkstra since the crossover and mutation operations of the GA may become less efficient. Experimentations also show that the quality of final solutions in our approach depends closely on the graph density. In dense graph, our algorithm is more likely to find the optimal solution. However, in sparse graph our approach usually fail to find the optimal path.

One reason to explain that fact lies in the close relation between the graph density and the performance of the VNS method used in our approach. As before mentioned, the construction of the two neighboring structure of the VNS method is done thanks to a preprocessing step that searches for an alternative two-edges-path to reach the end point of an edge from its starting point. Therefore, in dense graph, the probability that each solution in the population has better neighbor solutions increases gradually. Consequently, the algorithm's chance to overcome a local optimum and visits new regions in the search space increases dramatically.

To illustrate that fact, we test our algorithm over complete graph instances, which have a density of one. We realized that the algorithm's chance to find the optimal path is more than 98%. However, in graphs with low density, our algorithm's average gap to the optimality increases gradually (~5%). Finally, we have remarked that in complete graphs, the number of individuals in a population may decrease to two. In another term, a population with only two individuals may be sufficient for efficiently finding the optimal solution. Moreover, the algorithm can proceed with just two paths as initial solutions.

Table 6. Performing Single Point Crossover

Graph			Running time of exact approaches			Our Algorithm			GA		VNS	
TYPE	N	M	DIJKSTRA	IP		TIME	GAP (%)	SPEED	GAP (%)	SPEED	GAP (%)	SPEED
				CPLEX	GUROBI							
R	100	10000	14,90	949,87	599,99	1,68	0	8	3.14	10	10	41
			48,86	1441,19	507,34	0,60	0	81	5.2	15	27	31
			7,40	265,51	519,67	0,74	0	10	15.3	10	28	60
			2,94	1009,51	563,97	0,89	0	3	6.14	17	24	16
			2,42	369,62	525,92	0,65	0	3	8.64	7	18	49
R	1000	100000	194,29	7773,63	5767,28	2,37	0	81	19.4	16	25	52
			249,24	7229,51	5149,81	0,95	2.16	262	14.1	16	12	39
			54,13	7315,10	6081,39	1,23	1.12	44	14.3	4	12	68
			53,05	7551,22	5491,86	1,00	0.27	53	3.23	8	14	30
			53,81	7820,56	5907,35	1,10	0	48	17.9	4	18	84
	25000	500000	797,66	153502,72	29839,36	3,13	0	254	8.34	18	17	28

R			814,44	96358,00	18549,06	2,40	0	339	12.5	13	13	91
			578,03	83066,09	17961,71	1,87	0	309	20.1	14	26	60
			492,50	84785,43	18112,86	1,50	4.08	328	20.4	17	12	28
			538,94	87916,22	19438,90	2,53	0	213	12.4	11	24	28
C	50	2450	3,28	49,72	24,18	0,30	0	11	7.78	3	23	23
			2,40	59,33	11,08	0,25	0	9	16.4	5	10	51
			3,01	53,61	20,69	0,33	0	9	18.5	10	21	84
			1,52	39,66	9,37	0,16	0	9	8.23	17	12	11
			1,61	38,26	18,78	0,18	0	8	8.34	19	10	91
C	150	22350	15,96	440,10	153,98	0,53	0	30	3.34	8	27	50
			18,94	386,89	169,83	0,40	0	47	15.3	14	12	73
			5,19	418,59	216,41	0,50	0	10	12.6	16	10	29
			2,84	401,78	195,37	0,39	0	7	19.2	11	30	76
			2,26	404,61	143,91	0,37	0	6	20.6	8	16	21
C	500	249500	12,53	2485,40	2451,00	0,27	1.01	45	4.56	10	24	68
			12,87	5972,23	1514,43	0,29	0.11	44	16.7	10	22	95
			28,78	2689,79	2686,63	1,03	0	28	12.3	19	22	80
			25,05	2481,10	1189,08	0,90	0	28	20.6	16	14	11
			23,84	2447,22	2225,44	1,27	0	19	14.7	20	29	35
D	321271	800174	164,78	70794,05	223863,71	6,36	0.16	25	3.2	3	24	63
			222,42	81316,29	1073923,47	12,77	0.01	17	7.77	12	27	22
			166,48	74617,73	551245,39	13,83	1.13	12	17.5	8	21	33
			86,88	26491,70	27992,74	1,57	0	55	6.3	4	27	66
			213,79	81790,95	451932,16	17,88	0.19	12	22.3	9	10	68
D	1207946	2840210	2486,35	716962,02	169921,86	83,23	0.18	30	30.1	58	10	41
			1961,94	728352,15	2922319,64	126,26	1.03	16	15.2	15	27	31
			2150,12	772732,06	8993880,24	102,96	0.9	21	20.3	10	28	60
			1040,67	621584,55	9336397,10	40,96	0	25	6.2	17	24	16
			1330,92	586573,61	75536,35	29,49	4.15	45	8.5	7	18	49
D	1890816	4657744	4650,82	-	-	129,40	2.95	36	19.6	16	25	52
			5615,06	-	-	134,30	3.13	42	14.2	16	12	39
			5088,58	-	-	114,94	0	44	14.7	4	12	68
			5984,08	-	-	123,27	2.29	49	3.2	8	14	30
			6214,16	-	-	115,12	0.6	54	3.6	4	18	84

6. Discussions

We presented and analyzed in the previous section experimental results done over some graph instances. However, we only evaluated our approach regarding two criteria: the time performance (speed) and the solutions quality (GAP). In this section, we want to discuss another criterion to evaluate a meta-heuristic, which is the flexibility. Measuring the flexibility of a method is not straightforward. Indeed, there is no standard formulation to say whether this method is flexible or not. Nevertheless, we can get few hints about the flexibility level of one method by analyzing its capacity to respond to additional problem constraints.

After analyzing our approach, we have noticed that the proposed algorithm is not uniquely dedicated to solve the one-to-one version of SPP. It has however the ability to cope with other shortest path variants. For instance, our approach can be adapted for computing multi-modal or even multi-criteria shortest paths thanks to the capacity of the

selected meta-heuristics in handling such additional requirements. Besides, we assumed in this paper that the edges' weight are static. However, in many real life applications such as routing in road or computer networks, the weight of an arc is not always the same. It may change over time according to some circumstances. Therefore, a flexible SPP algorithm should not only deal with static graphs. Further aspects should be considered such as the dynamics and stochasticity. In contrast to conventional methods and speed-up techniques, our method is easily adapted to deal with such complex requirements. For instance, our algorithm can deal with stochastic networks by updating the list of neighborhood solutions belonging to the affected segments.

Moreover, the proposed method will result in a set of feasible paths between two points. Consequently, we have several high-quality approximated solutions to reach a target. Therefore, in real world issues like routing in road networks we can say that our algorithm provide travelers with several high quality routes to reach their destinations. Travelers may therefore prefer not taking the best-found route but the route that meets other needs and preferences. Routing data packets in communication networks is another example that shows how much it is important of an algorithm to provide several high-quality solutions. Indeed, having several high-quality solutions to reach a target in data networks will give network administrators or even routing protocols several possibilities to accomplish the routing task. Therefore, solutions can be selected in such a way the degree of congestion decreases.

Furthermore, one shortcoming of many SPP algorithms lies in their inability to handle negative edge weights. For instance, the correctness of Dijkstra's algorithm requires the edge weights to be all positive or 0. On the contrary, negative weights do not affect the performance of our approach since the way we construct the neighborhood solutions in the VNS method is not affected by the existence of negative values. Therefore, our method is likely to be used in a wide range of routing applications.

Additionally and in contrast to traditional SPP algorithms, meta-heuristic and precisely memetic algorithm show more flexibility to handle additional problem's constraints. For instance, our algorithm may handle multi-criteria shortest paths by modifying the crossovers and mutations operations. In conclusion then, the proposed method is more flexible than traditional methods since it can cope with a wide range of new problems and constraints.

7. Conclusion

We have addressed in our study the challenge of computing the one-to-one shortest path. We proposed a hybrid meta-heuristics and we compared it with two traditional shortest path algorithms (Dijkstra and IP) as well as with two other meta-heuristics (GA and VNS).

The proposed algorithm firstly accomplishes a double search algorithm in order to get initial feasible paths; it enhances then the quality of initial generated paths thanks to the VNS method. The neighboring structures of this latter are constructed thanks a preprocessing step done during the generation of the graph. After enhancing the initial solutions, crossovers and mutations operations are performed in the goal of improving the solutions. The VNS method is also used as a mutation operation.

The results showed that combination of meta-heuristics are good candidates to solve the SPP and they result in very near optimal solutions if they are correctly adapted to the problem. In contrast to the other meta-heuristics, our approach has resulted in a good tradeoff between the solution quality and the average speed. The metaheuristics GA and VNS outperforms our approach and the exact approaches in terms of running time. However, they usually provide solutions far from the optimal solution.

Our proposed method has not only the ability to provide high-quality solutions for the one-to-one version of spp within reasonable computational time. It can also deal with additional problem's constraints thanks to the capacity of the selected metaheuristics in dealing with such issues. From one side, we benefit from the power of GA in exploiting a solution space, and from the other side, we capitalize on VNS to escape from local optima and visits distant solution regions. Finally, several works have been planned to be accomplished in the near future. We will try first to establish a full sensitivity analysis, as well as, a statistical test to assess the performance of the proposed approach regarding variations on the algorithm's parameters. We will then start to apply such efficient combination of meta-heuristics for computing multi-objective shortest paths in a multimodal transportation system that has the capacity to evolve and change over time.

Acknowledgements

This research work has been carried out in the framework of the Technological Research Institute SystemX, and therefore granted with public funds within the scope of the French Program “Investissements d’Avenir”..

References

[1] Dijkstra, Edsger W. "A note on two problems in connexion with graphs."Numerische mathematik 1.1 (1959): 269-271.
[2] Bellman, Richard. On a routing problem. No. RAND-P-1000. RAND CORP SANTA MONICA CA, 1956.
[3] Hougardy, Stefan. "The Floyd–Warshall algorithm on graphs with negative cycles." Information Processing Letters 110.8 (2010): 279-281..
[4] Zeng, W., and R. L. Church. "Finding shortest paths on real road networks: the case for A*." International Journal of Geographical Information Science 23.4 (2009): 531-543.
[5] Hilger, Moritz, et al. "Fast point-to-point shortest path computations with arc-flags." The Shortest Path Problem: Ninth DIMACS Implementation Challenge74 (2009): 41-72.
[6] Davis, Lawrence, ed. Handbook of genetic algorithms. Vol. 115. New York: Van Nostrand Reinhold, 1991.
[7] Dorigo, Marco, and Mauro Birattari. "Ant colony optimization." Encyclopedia of Machine Learning. Springer US, 2010. 36-39.
[8] Kennedy, James. "Particle swarm optimization." Encyclopedia of Machine Learning. Springer US, 2010. 760-766.
[9] Behzadi, Saeed, and ALIA ALESHEIKH. "Developing a Genetic Algorithm for Solving Shortest Path Problem." NEW ASPECTS OF URBAN PLANNING AND TRANSPORTATION (2008).
[10] Gonen, Bilal. "Genetic Algorithm finding the shortest path in Networks." Reno: University of Nevada (2006).
[11] Kumar, Dr Rakesh, and Mahesh Kumar. "Exploring Genetic algorithm for shortest path optimization in data networks." Global Journal of Computer Science and Technology 10.11 (2010).
[12] Mohemmed, Ammar W., Nirod Chandra Sahoo, and Tan Kim Geok. "A new particle swarm optimization based algorithm for solving shortest-paths tree problem." Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. IEEE, 2007.
[13] Woeginger, Gerhard J. "Exact algorithms for NP-hard problems: A survey."Combinatorial Optimization—Eureka, You Shrink!. Springer Berlin Heidelberg, 2003. 185-207.Wolsey, Laurence A. Integer programming. Vol. 42. New York: Wiley, 1998.
[14] Gogna, Anupriya, and Akash Tayal. "Metaheuristics: review and application."Journal of Experimental & Theoretical Artificial Intelligence 25.4 (2013): 503-526.
[15] Cantú-Paz, Erick. "A summary of research on parallel genetic algorithms." (1995).
[16] Mladenović, Nenad, and Pierre Hansen. "Variable neighborhood search."Computers & Operations Research 24.11 (1997): 1097-1100.
[17] Hoffman, Karla L., Manfred Padberg, and Giovanni Rinaldi. "Traveling salesman problem." Encyclopedia of Operations Research and Management Science. Springer US, 2013. 1573-1578.
[18] Lawler, Eugene L., and David E. Wood. "Branch-and-bound methods: A survey." Operations research 14.4 (1966): 699-719.
[19] Raidl, Günther R. "A unified view on hybrid metaheuristics." Hybrid Metaheuristics. Springer Berlin Heidelberg, 2006. 1-12.
[20] Dowsland, Kathryn A., and Jonathan M. Thompson. "Simulated annealing."Handbook of Natural Computing. Springer Berlin Heidelberg, 2012. 1623-1655.
[21] Glover, Fred, and Manuel Laguna. Tabu Search∗. Springer New York, 2013.
[22] Bäck, Thomas, and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization." Evolutionary computation 1.1 (1993): 1-23..
[23] Hansen, Pierre, and Nenad Mladenović. Variable neighborhood search. Springer US, 2005.
[24] Talbi, E-G. "A taxonomy of hybrid metaheuristics." Journal of heuristics 8.5 (2002): 541-564.
[25] Moscato, Pablo, Carlos Cotta, and Alexandre Mendes. "Memetic algorithms."New optimization techniques in engineering. Springer Berlin Heidelberg, 2004. 53-85.