Genetic algorithm combined with variable neighborhood search to solve the one-to-one shortest path problem

Omar Dib[1], Alexandre Caminada[2], Marie-Ange Manier[3]

[1] Technological research institute IRT SystemX
8, avenue de la vauve, 91120 PALAISEAU
omar.dib@irt-systemx.fr

[2] University of technology of belfort-montbéliard
OPERA– UTBM, 90010 Belfort Cedex France
alexandre.caminada@utbm.fr

[3] University of technology of belfort-montbéliard
OPERA– UTBM, 90010 Belfort Cedex France
marie-ange.manier@utbm.fr

Abstract

The shortest path problem (SPP) is widely applied in various fields of application such as transportation, telecommunication and computer networks. It aims at determining a path between two points with the minimum associated/allocated resource (distance, time, cost…) in a directed or undirected graph. Several algorithms have been proposed to solve the standard SPP such as Dijkstra, Bellman-Ford. However, such approaches become less performant or even inapplicable when the size of the network becomes very large or when additional constraints are added such as dynamic arcs weight, multi-criteria paths optimization. To address these issues, researchers have thought about applying meta-heuristics to solve the standard SPP and its different variants. We introduce in this paper a novel approach to solve the one-to-one version of SPP. Our method is based on a combination of Genetic Algorithm (GA) and Variable Neighborhood Search (VNS). We compare our method with two exact algorithms Dijkstra and Integer Programming (IP). We made experimentations using random generated, complete and real graph instances. In most case studies, numerical results show that the average speed of our algorithm is 20-times faster than Dijkstra and more than 1000-times compared to IP. While the algorithm of Dijkstra and the IP techniques provide optimal solutions, the average gap to the optimality of our approximate approach is 5%.

1 Introduction

Computing shortest paths is one of the classical and well-studied problems in combinatorial optimization, with various applications in theory and practice. Numerous real-world applications such as routing in road networks, transferring data in computer networks etc. have encouraged the study of the shortest path problem for more than 60 years. 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 SPP, which we address in this study, is defined as that of finding a path with the minimum-length (cost) path between a given pair of nodes. Several algorithms have been proposed since 1956 to compute shortest paths. The standard solution to the one-to-one SPP is Dijkstra's algorithm [1]. To deal with negative edge weight, we have the Bellman-Ford algorithm [2]. Floyd-Warshall [3] also proposed an algorithm to find shortest paths between all pairs of points in a weighted graph with positive or negative edge weights. SPP was also solved using Integer programming (IP) techniques [4], but its running time is usually higher than standard algorithms. Although the aforementioned algorithms can optimally solve the SPP, they exhibit, however, unacceptably high computational complexity for real-time communications in large-scale graphs [5]. Extensive researches have therefore been conducted to accelerate traditional algorithms using speed up techniques such as in [6] and [7]. Standard algorithms may also become less performant or even inapplicable when considering additional problems' requirements such as multi-objective shortest paths, dynamic arcs' weights. To handle these limitations, researchers have thought about applying meta-heuristics [8] to find optimal or near optimal paths within a reasonable computational time. Meta-heuristics also offer high performances and flexibilities to support several categories of optimization (single criteria, multi-criteria) whether they are in static, dynamic or even stochastic graphs.

Evolutionary algorithms [9] such as, Genetic Algorithms (GAs) [10], Particle Swarm Optimization (PSO) [11] Ant Colony Optimization (ACO) [12] are one of the meta-heuristics, which have been used for solving SPPs. For instance, references [13] and [14] worked with evolutionary algorithms to compute single source shortest paths using single-objective fitness. Reference [15] proposed a genetic algorithm to find shortest paths in computer networks. Moreover, reference [16] used a genetic algorithm to solve the routing issue in road networks. References [17] and [18] also worked with genetic algorithms to find shortest paths in data networks. Reference [19] proposed effective crossover operators for the all-pairs SPP. Reference [20] combined PSO with local search and velocity re-initialization for computing shortest paths in computer networks. Although these works are very interesting, we have remarked that such new approaches were only tested on small network instances (ex: 100 nodes). Therefore, we have endeavored to make experimentations over large-scale networks. The rest of this paper is organized as follows. In next section, we present the meta-heuristics GA and VNS. Section 3 is devoted to present our approach. We present and discuss in Sections 4 and 5 our experimental results. Finally, section 6 concludes this paper and mentions some future works.

2 Hybrid Metaheuristics

Hybrid meta-heuristics [21] have received considerable interest in combinatorial optimization. The main motivation for the hybridization process is to obtain better performance by exploiting and combining advantages of the individual pure strategies. 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.

2.1 Genetic Algorithm (GA)

GAs are search methods based on the evolutionary ideas of natural selection and genetics [10]. Unlike traditional search techniques, GAs are capable of avoiding randomness search by intelligently exploiting historical information that guide the search process towards regions of better performance within the solution space. After an initial population is randomly generated, GAs perform through three main steps: Selection, Crossover, and Mutation [22].

2.2 Variable Neighborhood Search (VNS)

VNS is a single-solution based meta-heuristic proposed by Hansen and Meladenovic in 1997 [23]. It is used nowadays for solving a various range of combinatorial and global optimization problems ((linear, nonlinear, continuous, discrete etc. . .). VNS's strengths lies in its ability to systematically explore several distant neighborhoods by making a perturbation move (called shaking in the VNS literature) to escape from local minima [24]. Therefore, VNS is more likely to prevent the optimization process from rapidly falling into local optima. Unlike traditional single-search based meta-heuristics such as tabu search [25] and simulated annealing [26], VNS algorithm structure is very simple and does not require many parameters.

3 Proposed Approach

As aforementioned, the main contribution of this paper is to combine two meta-heuristics (GA & VNS) in order to solve the one to one SPP. As GAs work, our method maintains a population of solutions at each iteration. The initial population is generated thanks to a double-search algorithm that we applied to get a set of initial feasible paths between two nodes. After getting initial paths, we enhance the first population's individuals using VNS. We decided with improving initial solutions since their quality would possibly help the algorithm finding better solutions. After that, we evaluate individuals. Our fitness function is the sum of the weights of each edge included in a path. After evaluation, we repeatedly perform some genetic operations in order to improve our solutions. We begin with the crossover operation. We have used variable-point (single/two points) crossover with a probability of 0.9. The selection operation is done using roulette wheel technique. After crossover, we perform, with low probability, a special mutation operation based on VNS. Thanks to this technique, our algorithm will have more

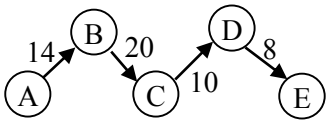
chances to explore new regions of the search space as well as efficiently performing exploiting process. The selection, crossover and mutation operations are repeated until the algorithm reaches our stopping criteria. One point worth mentioning is that the probabilities of genetic operators are not chosen arbitrary. Experimentations have proven that our algorithm results in better performance when the probability of crossover is very high compared to the probability of mutation.

Algorithm 1. Steps of our algorithm

- Create Initial Population
- Apply VNS
- Evaluation
- Repeat
  - Selection (Roulette wheel)
  - Crossover(one/two-points; probability:0.9)
  - Mutation (VNS;Probability:0.1)
- Until stopping criteria are met

3.1 Encoding

As in all GAs, the encoding phase of individuals is a crucial step. It may have a significant impact on the algorithm performance. To encode an individual, we have used the permutation encoding. Each individual is represented by an array of integers representing the identifiers of edges including in the path. The dimension of chromosomes is not fixed, since the number of edges in each chromosome may change.



Chromosome :	*AB*	*BC*	*CD*	*DE*

Figure 1. Encoding of chromosomes

3.2 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.

Finally, as the algorithm performs, there is a chance that two identical paths are found between 'S' and 'T'. To overcome this problem, the algorithm stores the intersection edges found during the execution. The algorithm disregard then a path if the intersection edge has been already used.

3.3 Applying VNS

The VNS used in our algorithm is based on two-neighborhood structures. To construct those latters, we accomplish 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.

Example: If we take the edge (AC) in Fig.2, 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, there is an alternative path to reach the node C from A, which is in this case the path (AB, BC).

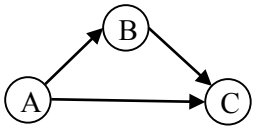


Figure 2. Constructing neighboring structures

Once we find such common nodes, we can start constructing our two neighboring structures.

✓ 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 simple edge (AC) in any path. Our second neighboring structure can then be seen as a list containing an edge that will replace a path formed by two edges.

Example: Fig.3 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 from the second replacement structure.

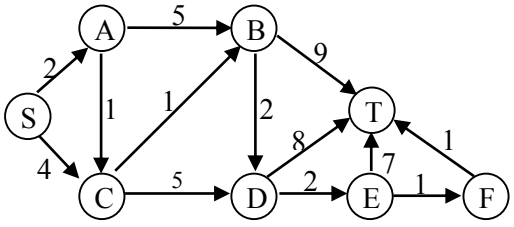


Figure 3. Performing VNS over an individual

After performing the preprocessing operation over Fig.3, we will end up with two replacement lists that represent our neighboring structures.

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

Table 1. Items in the first replacement list

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

Table 2. Items from the second replacement list

After defining our neighboring structures, we want to explain now how the algorithm uses such information in order to improve the quality of an individual. To do so, Let us assume that the path *P*= (SC, CB, BD, DT) (Fig.3), is an individual representing a solution to go from S to T. The length of *P* is 15. The algorithm applies the VNS over *P* as follows:

✓ Initially, the algorithm uses the first neighboring structure to perform local refinements over an individual. It goes through each edge in the path and tries to detect if there is a replacement for that edge. In our example, the individual *P* can be improved by substituting the edge (SC) by the path (SA, AC). The new path generated is then (SA, AC, CB, BD, DT) and its length is 14. As can be noticed, the new path does not contain edges included in the first replacement structure so the algorithm switches to the second neighborhood structure.

✓ The algorithm examines then each two successive edges and searches for a replacement. In our example, we remark that the path (BD, DT) including in *P* is in the second replacement structure. Hence, the algorithm replaces it by the edge (BT). The new path generated is (SA, AC, CB, BT) and its length is improved to 13.

✓ As can be remarked, the initial path *P* has been successfully improved. However, we still do not attend the optimal path. Such shortcoming is stemmed from the fact that our neighboring structures do not allow non-improving replacements. To overcome such challenge, we applied some enhancement operations on the replacements structures.

✓ Enhancement of neighborhood structures would probably improves the quality of neighbor solutions. To ameliorate our structures, we enhance the quality of elements existing in the first structure using information from itself and from the second structure. For instance, the path (DE, ET) included in the second structure can be replaced by the edge (DT). However, if we carefully scrutinize, we can notice that the edge (ET) is 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 the optimal solution for our example (SA, AC, CB, BD, DE, EF, FT).

Our VNS method is based on two-neighboring structures. At each time our algorithm gets trapped in 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 structures will possibly enhance the chance of the algorithm to find the best path. However, that might increases the time to accomplish the preprocessing operation. In addition, we chose randomly to apply the first structure improvement before the second one. In fact, detecting which structure should be applied at which order has been always an issue for VNS method. Further works will be done in the near future to deal with such challenges.

3.4 Fitness Evaluation

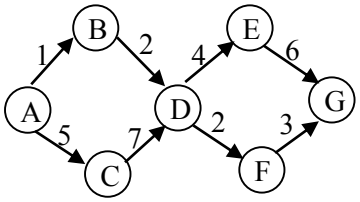
The fitness function used by our algorithm is the sum of edges' weights including in a path. It is the same as our objective function. The result is a non-negative number that represents the path length. The cost to get from one node to another is stored as a parameter into the edge entity.

3.5 Crossover

To accomplish the crossover process, we sort firstly the individuals in the population according to their fitness values. We then repeatedly select the best two individuals to be the crossover parents. We then try to find two common nodes to be the crossover points (two-point crossover). If we cannot find two common nodes, we only look for one node (single-crossover). If we cannot perform both single/two

point (s) crossovers, we replace one parent by its successor individual from the sorted population.

Once crossover point (s) are detected, we take portions from each parent and we recombine them in order to produce new individuals (offspring). By doing this, our algorithm will have the chance to visit new regions within the search space.



P_2:F=17	AC	C**D**	**D**F	FG
P_1:F=13	AB	B**D**	**D**E	EG

I_2:F=22	AC	CD	DE	EG
I_1:F=8	AB	BD	DF	FG

Figure 4. Example of a single point Crossover

One point worth mentioning is that after our crossover operation, we do not care about the feasibility of the new individuals. We will always end up with a feasible solution from the source node to the target. Therefore, the algorithm does not lose time to test the validity of the offsprings nor to perform some additional operations to repair the infeasible solutions.

We have remarked during experimentations that the result of crossover are sometimes individuals that already exist in the population. Hence, the algorithm might check the same individual more than one time. To palliate this problem, we have proposed two techniques.

✓ The first one is to add an additional constraint on the way we choose our crossover points. Not any common node between two individuals should be considered as a crossover point. When two parents have, the same sequence of edges from the source node to the crossover point, or from the crossover node to the destination node, this means that the crossover operation will result in two individuals that are identical to their parents. The crossover operation will therefore fail to provide new paths.

✓ Another way to ensure the uniqueness of individuals in the population is to compare each offspring generated with all the individuals and eliminate then the redundant one. However, this may lead to a significant running time especially when the population size is very large. Thus, we decide with using the first solution.

3.6 Mutation (VNS)

Crossover operation may produce degenerate population. The algorithm may therefore get stuck in local minima. To overcome this problem, we perform the mutation operation. We have chosen the VNS as a mutation operator. That is, we apply VNS with low probability over the population's individuals. By doing so, we guide the algorithm towards new regions within our search space. We therefore increase the algorithm's chance to find better solutions. We did apply other mutation techniques such as order changing, but we have realized that the mutation in this case may result in invalid paths. An additional process should therefore be applied to reform infeasible paths. As a result, the mutation computational time will increase. We decided so against using such traditional mutation techniques.

3.7 Stopping Criteria

Unlike classical exact resolution methods, meta-heuristics do not guarantee the finding of optimal solutions. Additional terminating conditions should therefore be added to allow the convergence of the algorithm. Maximum number of generations, fixed execution time, and no modifications in population individuals are considered as algorithm stopping criteria. Our algorithm terminates in two cases:

✓ If the individual with the best fitness is repeated for longer time. We have used 100 generations as a number to ensure a fixed state in the population.

✓ If the maximum number of generations is reached, (say 500).

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 numbers used when parameterizing the stopping criteria for our algorithm.

4 Experimental Results

We have done experimentations over 50 different graphs instances (from small to large size) generated thanks to a graph generator that we developed using Java. We run the used algorithms and solvers on an Intel core I5 machine of 8 GB RAM. For simplicity, we present in table 3 nine graph instances; each belongs to one of these three categories: Random, Complete and DIMACS Graph. For each instance, we choose five-times two random points to construct our origin-destination queries. We compared the running time of our approach with two other exact algorithms (Dijkstra and IP). We implemented Dijkstra using a priority queue and we solved the IP generated using two solvers: Cplex 12.6 and Gurobi 6.0. 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 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 is 20-times faster than Dijkstra and more than 1000-times compared to IP. Moreover, results show that the time spent to solve the integer programs 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. Experimentations also show that the quality of final solutions depends closely on the graph density. In dense graphs, the algorithm is more likely to find the optimal solution. However, in sparse graphs our approach usually fails to find an optimal path. One reason to explain that fact lies in the close relation between the graph density and the performance of the VNS. 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 graphs, 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 better 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 an optimal path is more than 98%. However, in graphs with low density, our algorithm's average gap to the optimality increases to (~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. We will discuss in future works the effects of the population size over our algorithm's performance.

Graph			**Running Time (msec)**				**Average**	
Type	Nodes	Edges	Dijkstra	Ip		Our Algorithm	Gap (%)	Speed
				Cplex	Gurobi			
RANDOM	100	10000	14,90	949,87	599,99	1,68	0	8
			48,86	1441,19	507,34	0,60	0	81
			7,40	265,51	519,67	0,74	0	10
			2,94	1009,51	563,97	0,89	0	3
			2,42	369,62	525,92	0,65	0	3
RANDOM	1000	100000	194,29	7773,63	5767,28	2,37	0	81
			249,24	7229,51	5149,81	0,95	2.16	262
			54,13	7315,10	6081,39	1,23	1.12	44
			53,05	7551,22	5491,86	1,00	0.27	53
			53,81	7820,56	5907,35	1,10	0	48
RANDOM	25000	500000	797,66	153502,72	29839,36	3,13	0	254
			814,44	96358,00	18549,06	2,40	0	339
			578,03	83066,09	17961,71	1,87	0	309
			492,50	84785,43	18112,86	1,50	4.08	328
			538,94	87916,22	19438,90	2,53	0	213

COMPLETE	50	2450	3,28	49,72	24,18	0,30	0	11
			2,40	59,33	11,08	0,25	0	9
			3,01	53,61	20,69	0,33	0	9
			1,52	39,66	9,37	0,16	0	9
			1,61	38,26	18,78	0,18	0	8
COMPLETE	150	22350	15,96	440,10	153,98	0,53	0	30
			18,94	386,89	169,83	0,40	0	47
			5,19	418,59	216,41	0,50	0	10
			2,84	401,78	195,37	0,39	0	7
			2,26	404,61	143,91	0,37	0	6
COMPLETE	500	249500	12,53	2485,40	2451,00	0,27	1.01	45
			12,87	5972,23	1514,43	0,29	0.11	44
			28,78	2689,79	2686,63	1,03	0	28
			25,05	2481,10	1189,08	0,90	0	28
			23,84	2447,22	2225,44	1,27	0	19
DIMACS	321271	800174	164,78	70794,05	223863,71	6,36	0.16	25
			222,42	81316,29	1073923,47	12,77	0.01	17
			166,48	74617,73	551245,39	13,83	1.13	12
			86,88	26491,70	27992,74	1,57	0	55
			213,79	81790,95	451932,16	17,88	0.19	12
DIMACS	1207946	2840210	2486,35	716962,02	169921,86	83,23	0.18	30
			1961,94	728352,15	2922319,64	126,26	1.03	16
			2150,12	772732,06	8993880,24	102,96	0.9	21
			1040,67	621584,55	9336397,105	40,96	0	25
			1330,92	586573,61	75536,39705	29,49	4.15	45
DIMACS	1890816	4657744	4650,82	-	-	129,40	2.95	36
			5615,06	-	-	134,30	3.13	42
			5088,58	-	-	114,94	0	44
			5984,08	-	-	123,27	2.29	49
			6214,16	-	-	115,12	0.6	54

Table 3. Experimental results

5 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 our approach, which is the flexibility. Measuring the flexibility of a method is not straightforward. There is no standard formulation to say whether a 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. Further papers will be published in the near future to present our results and explain how our approach can meet such requirements. Besides, we worked in this paper with static edges' weight. However, in many real world applications such as routing in road or computer networks, the weight of an arc is not always fixed. It may change over time due to some circumstances such as congestions, perturbations etc. Therefore, a flexible shortest path algorithm should not only consider a network as a static graph. Further aspects should be handled such as the dynamics and stochasticity. Unlike conventional algorithms like Dijkstra, our algorithm can use the VNS method to deal with stochasticity by simply 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 instead of only one solution. Consequently, we have several high-quality approximated solutions to reach a target. Therefore, in real world issue 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 in data networks is another example to show how important is for a routing algorithm to provide several high quality routes. To route data packets for example, our algorithm provide several high-quality routes. Therefore, network administrators or even routing protocols may choose some routes in such a way the degree of congestion decreases. Furthermore, one shortcoming of many conventional shortest path 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. In contrast to such algorithms, negative weights do not affect the performance nor the correctness 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. In addition, the meta-heuristics used in this study are known as powerful methods to solve a various range of optimization problems. For instance, our algorithm can be easily extended to handle multi-objective shortest paths. Few modifications in the genetic operators would handle multi-criteria optimization. As discussed before, computing shortest paths shouldn't only be based on one metric. An ideal routing process should optimize several metrics at the same time. 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.

6 Conclusion

We have addressed in our study the challenge of computing the one-to-one SPP. We proposed a hybrid meta-heuristic in which we use the single-solution based meta-heuristic (VNS) inside the population-based method (GA). We compared our algorithm with two traditional shortest path algorithms (Dijkstra and IP). Experimental results showed that the running time of traditional algorithms increases rapidly with the size of the graph. Therefore, using such algorithms for computing shortest paths in large-scale networks within a reasonable computational time may become impossible. However, our approach provide optimal or near optimal solutions in a reasonable time even when the network's size increases. Furthermore, the proposed combination has not only the ability to provide high-quality solutions for the one-to-one version of spp. It can however deal with additional problem's constraints. From one side, we benefit from the flexibility and power of GA in exploiting a large solution space, and from the other side, we capitalize on VNS to escape from local optima and visits distant solution regions. As a result, combining meta-heuristics and especially those who belong to different categories (single/population based) is a powerful way to solve the standard SPP and its different variants. However, further attentions should be given to the meta-heuristics' parameters to reach better performance levels. Currently, we are working on applying our method for computing shortest paths in a large-scale multimodal transportation network. We are mainly considering railway, bus and pedestrian networks. The results obtained so far are very encouraging and they prove the efficiency and flexibility of our method. Finally, we have planned to accomplish several works in the near future. We planned to apply our approach to solve the multi-objective shortest path issue in a multimodal transportation system that has the dynamics and stochastic aspects.

Acknowledgments

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] Garfinkel, Robert S., and George L.Nemhauser. Integer programming. Vol. 4. New York: Wiley, 1972.

[5] Ahn, Chang Wook, and Rudrapatna S. Ramakrishna. "A genetic algorithm for shortest path routing problem and the sizing of populations." Evolutionary Computation, IEEE Transactions on 6.6 (2002): 566-579.

[6] Delling, Daniel. Engineering and augmenting route planning algorithms. Diss. Karlsruhe Institute of Technology, 2009.

[7] H. Bast, S. Funke, P. Sanders, D. Schultes (2007): Fast routing in road networks with transit nodes. Science 316 (5824):566-566.

[8] Blum, Christian, and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison." ACM Computing Surveys (CSUR) 35.3 (2003): 268-308.

[9] Bäck, Thomas, and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization." Evolutionary Computation 1.1 (1993): 1-23.

[10] Davis, Lawrence, ed. Handbook of genetic algorithms. Vol. 115. New York: Van Nostrand Reinhold, 1991.

[11] Kennedy, James. "Particle swarm optimization." Encyclopedia of Machine Learning. Springer US, 2010. 760-766.

[12] Dorigo, Marco, and Mauro Birattari. "Ant colony optimization." Encyclopedia of Machine Learning. Springer US, 2010. 36-39.

[13] Surender Baswana, Somenath Biswas, Benjamin Doerr, Tobias Friedrich, Piyush P. Kurur, Frank-Neumann: Computing single source shortest paths using single-objective fitness. FOGA 2009: 59-66

[14] Benjamin Doerr, Edda Happ, Christian Klein: Tight Analysis of the (1+1)-EA for the Single Source Shortest Path Problem. Evolutionary Computation 19(4): 673-691 (2011)

[15] Kumar, Rakesh, and Mahesh Kumar. "Reliable and Efficient Routing Using Adaptive Genetic Algorithm in Packet Switched Networks." IJCSI International Journal of Computer Science Issues 9.1 (2012): 1694-0814.

[16] Gonen, Bilal. "Genetic algorithm finding the shortest path in networks". Reno, Nevada 89502, CiteSeerX, 2004.

[17] Behzadi, Saeed, and ALIA ALESHEIKH. "Developing a Genetic Algorithm for Solving Shortest Path Problem." NEW ASPECTS OF URBAN PLANNING AND TRANSPORTATION (2008).

[18] 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).

[19] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, Madeleine Theile: More Effective Crossover Operators for the All-Pairs Shortest Path Problem. PPSN (1) 2010: 184-193

[20] 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

[21] Talbi, E-G. "A taxonomy of hybrid metaheuristics." Journal of heuristics 8.5 (2002): 541-564.

[22] Mitchell, Melanie. An introduction to genetic algorithms. MIT press, 1998.

[23] Mladenović, Nenad, and Pierre Hansen. "Variable neighborhood search."Computers & Operations Research 24.11 (1997): 1097-1100.

[24] Hansen, Pierre, and Nenad Mladenović. Developments of variable neighborhood search. Springer US, 2002.

[25] Glover, Fred, and Manuel Laguna. Tabu Search∗. Springer New York, 2013.

[26] Dowsland, Kathryn A., and Jonathan M. Thompson. "Simulated annealing."Handbook of Natural Computing. Springer Berlin Heidelberg, 2012. 1623-1655.