Initialization methods for the TSP with Time Windows using Variable Neighborhood Search

Christos Papalitsas, Konstantinos Giannakis,
Theodore Andronikos, and Dimitrios Theotokis
Ionian University
Department of Informatics
Corfu, Greece
Email: {c14papa, kgiann, andronikos, dtheo}@ionio.gr

Angelo Sifaleras
University of Macedonia
Department of Applied Informatics
Thessaloniki, Greece
Email: sifalera@uom.gr

***Abstract*—This paper presents a General Variable Search heuristic, trying to discover initial, feasible solution for the Travelling Salesman Problem with Time Windows. More specifically, we provide all relevant information regarding TSP-TW, Variable Neighborhood Search (VNS) and initialization methods, then we quote all related work in the direction on solving this NP-HARD problem, and at last we propose our new implementation for generating initial, feasible solution for the TSP-TW. We are thoroughly focused on the behavior of our main algorithm to different random-based or sorting-based initialization procedures of our main implemented algorithm. At last, we show experimentally that in some cases, the use of a sorting function as an initialization function in this algorithm did not work and never returned a feasible solution in some cases.**

I. Introduction

Our main research deals with the investigation of the Traveling Salesman Problem (TSP) and the presentation of the meta-heuristic method GVNS, a sub-category of VNS, which is used, inter alia, to solve this problem. The GVNS is an extension of VNS method which routinely performs the process of changing the neighborhood, both in descent for the local minimum, and to escape from the narrow regions ("valleys") containing them. The traveling salesman problem is used as a reference in many optimization methods and has numerous applications in many different areas such as Artificial Intelligence, Machine Learning, Software Technology etc. In this paper we refer particularly to the embodiment of the TSP problem, which uses some additional constraints such as time windows. This version is known as TSP-TW (Traveling Salesman Problem with Time Windows).

In this work we describe the GVNS meta-heuristic method, used for solving the TSP-TW problem and then a new implementation of GVNS method for this problem is presented, in the programming language JAVA. Furthermore, we try to conduct a comparative study leading to specific conclusions, about different versions of initialization procedures that we apply to our benchmark problems, before our main algorithm starts running in the direction to find feasible solutions. Finally, we apply this implementation to various benchmark problems and also to some new modified benchmark problems, and we present the results from a comparative study of sorting and random functions made.

The first use of the term "Traveling Salesman Problem" appeared around 1931-1932. But a century earlier, in 1832 a book was printed in Germany [1], which although dedicated to other issues, in the last chapter reaches the substance of the TSP problem: "With a suitable choice and route planning, you can often save so much time to make some suggestions... The most important aspect is to cover as many locations, without visit a location for the second time ..."[2]. In this manual, the TSP is expressed for the first time, including some examples of routes through Germany and Switzerland. However, this manual did not deal more in depth with this [1]. The TSP was expressed mathematically for the first time in the 19th century by Hamilton and Kirkman.

A cycle in a graph is a close path visiting each node exactly once and starts and ends in the same node. If a circle contains all the vertices of the graph, then it is called Hamiltonian [2]. TSP is the problem of finding the shortest Hamiltonian cycle. Conversely, the problem of deciding on whether a graph has a Hamiltonian cycle is a special case of the traveling salesman problem: If you assign to zero length edges of the graph (0) and each edge is missing, create an edge length of one (1), then we have a new graph, which is Hamiltonian cycle. The resolution of this particular case of TSP will be a Hamiltonian cycle for the new graph, which will have either total length zero (0), so it will be a Hamiltonian cycle for the original graph, or positive length, which demonstrates that there is no such cycle on the original graph.

This problem is NP-hard, with wide significance in various fields, such as operational research and theoretical computer science. TSP refers to finding the best way by which, we can visit all the cities, return to the starting point, and minimize the cost of tour. It can be set to a complete undirected graph G = (V, E), whether it is symmetrical or in a directed graph G = (V, A), if it is asymmetric. The set V = {1,2,3, ..., n} is the set of vertices, E = {(i, j): i, j ∈V, i $<$ j} is the set of either edges or arcs. By n we denote a number of cities (nodes). A cost matrix C = ($c_{i,j}$) is defined in E or A. This matrix satisfies the triangle inequality whenever $c_{i,j} \leq c_{i,k} + c_{k,j}$ for all c, j, k. In particular this is the case for the problems where the vertices are points $P_i = (X_i, Y_i)$ in a 2-d space, and $c_{i,j} = \sqrt{((X_i - X_i)^2 + (Y_i - Y_j)^2)}$ is the Euclidean distance. The triangle inequality is also satisfied if the $c_{i,j}$ is the length of the smallest path from i to j in the graph G [2].

Regarding the computational complexity of symmetric TSP, the total number of possible paths is equal to (n-1)!/2,

where n is the number of cities. TSP-TW is a variant of the TSP, except that cities (or customers) should be visited in a certain period (or "window"). This additional time constraint limits the search tree because all TSP solutions are not necessarily TSP-TW solutions. This time constraint makes the problem even more difficult in practice. The beautiful symmetry of TSP, where any permutation of cities is a feasible solution, is violated and even the search of feasible solutions is difficult [10]. Given a depot, a set of customers, the service time (i.e., the time that should be spent on client) and a time window (i.e., the start and the expiration time), the TSP-TW problem refers to finding a shortest path, which starts and ends at a given depot, since each customer has been only visited once before its expiry date.

The way we determine the depot is important for the TSP-TW, while this is not true for the general version of the TSP. The Travelling Salesman or agent is allowed to reach the customer before its ready time, but must wait. Obviously, there are paths that do not allow the agent to respect the deadlines of all customers. We call these paths infeasible, while many paths allow the agent to respect the deadlines of all customers called feasible paths. The cost of a path is the total distance traveled [5]. Given graph G = (V, A), wherein V = $\{1,2, ..., n\}$. Suppose 0 indicates a depot and A = (i, j): i, j $\in$ V $\cup$ $\{0\}$ is the set of arcs between clients. The cost of tour (traveling cost) from i to j is represented by c_{ij}, which includes both the customer service time i, and the time it takes to go from i to j. Each customer i is associated with a time window a_i, b_i, where a_i and b_i represent the readiness and maturity time, respectively. Therefore, the TSP-TW can be formulated as a problem of finding a Hamiltonian path, which starts and ends in the depot, satisfying all timing constraints and minimizing the total distance traveled [4].

II. Related Work

Regarding exact algorithms for TSP-TW, Langevin et al. introduce a flow formulation of two elements, which can be extended to the classic "makespan" problem. With the term "makespan" we mean the maximum processing time and integration, i.e. when all jobs have finished processing [5]. Dumas et al. [6] used a dynamic programming approach of elimination trials, which improved the performance and reduced the search space, both in advance and during its execution. Pesant et al. [7] proposed a branch-and-bound algorithm, using a programming model with restrictions. Similarly, Focacci et al. [8] proposed a hybrid approach, based on programming with constraints and optimization techniques [9].

The TSP-TW is NP-hard problem, since it is a special case of the well-known TSP. Thus, there is need for a heuristic, which is able to solve effectively realistic cases within a reasonable time. Carlton and Barnes [11] used a tabu search heuristic with static penalty function using infeasible solutions in the search. Gendreau et al. [12] proposed an insertion heuristic based on the heuristic GENIUS [13], which gradually builds the path to the construction phase and improves a refinement phase (based on sequential removal and reinsertion of nodes). Calvo [14] solved an assignment problem with an ad-hoc objective function and builds a feasible route, merging many such paths that have been found in a main route and then a 3-opt local search procedure is applied to improve the initial feasible solution. Ohlmann and Thomas [15] use a variant of simulated annealing, called compressed annealing, which relaxes the constraints of time window, incorporating a variable penalty method in a stochastic search process.

Two new heuristics were proposed by Lopez Ibanez [20] and Da Silva et al. [9]. In [4] the authors provide an interesting comparison of the results of these heuristics, as well as a new heuristic for TSP-TW. Currently, the three heuristics above can be considered to be the most important heuristics for the TSP-TW problem. Lopez Ibanez [16] propose a hybrid method which combines ant colony optimization with the radial search. These algorithms are called Beam-ACO algorithms and rely heavily on accurate and computationally inexpensive bounding information for differentiating between different solutions. Da Silva et al. [9] propose a two-stage heuristic, based on VNS. In the first step, a feasible solution is manufactured using the VNS, wherein the mixed linear, integer, objective function is represented as an infeasibility measurement. In the second stage, the heuristic improves feasible solution with a general version of VNS (GVNS).

III. Metaheuristic Procedures - VNS

A meta-heuristic is a high level heuristic, designed to find, create, or select a lower level heuristic (for example a local search algorithm), which can provide a pretty good solution to an optimization problem, particularly with missing or incomplete information, or with limited computing capacity [19]. According to the literature on the meta-heuristic optimization [18], the word "metaheuristics" was devised and proposed by Glover. Meta-heuristic processes can make some assumptions about the optimization problem to be solved and thus, they can be used for a wide variety of problems . Compared with exact methods, meta-heuristic procedures do not guarantee a global optimal solution for each category of problems [20].

Many meta-heuristic procedures apply some form of stochastic optimization. Thus, the generated solution depends on the set of random variables produced [19]. By searching in a large set of feasible solutions, the meta-heuristic procedures can often find good solutions with less computational effort than exact algorithms, iterative methods, or simple heuristic procedures [20]. Therefore, meta-heuristic procedures are useful approaches for optimization problems [19]. Numerous books and research articles have been published on meta-heuristic procedures [19], [20], [21], [22], [23]. The main literature on meta-heuristics procedures describes various empirical results based on experiments on computer algorithms. However, there are also some formal theoretical results, which mostly refer to the convergence and the possibility of finding the global optimal [20].

A. Variable Neighborhood Search

VNS, is a meta-heuristic method for solving a set of combinatorial problems and global optimization problems. It explores distant neighborhoods of the current incumbent solution and moves from there to a new solution if and only if there has been improvement. VNS manages a local search method, which is iteratively applied (iterated local search), to take the local optima of the preceding solutions in the neighborhood. This strategy is driven by three principles: 1) a

local minimum for a neighborhood structure cannot be a local minimum to a different neighborhood structure 2) a global minimum is a local minimum for all possible structures of the neighborhood, and 3) a local minimum is closely related to the total minimum for many classes of problems [17].

Meta-heuristic local search is performed by selecting an initial solution x, finding a descent direction of x, in a neighborhood $N(x)$ and moving towards the minimum of $f(x)$ in the $N(x)$ in the same direction. If there is no direction of the descent, then the heuristic stops; otherwise it is repeated. At each step, the neighborhood $N(x)$ of x has been fully explored [3]. VNS is designed for finding approximate solutions of discrete and continuous optimization problems and according to them, solving integer programming, mixed integer programming, nonlinear programming, etc. [3]. VNS was suggested by Mladenovic and Hansen [24], [25]. During the past years, several applications of VNS have been proposed in the literature [27], [28].

IV. Constructive phase of Initial Feasible Solution

In [10], Salvesbergh argues that, even the construction of a workable solution to the TSP-TW is an NP-hard problem. Gendreau et al. [12] attempt to construct feasible solutions using an insertion heuristic with backtracking. The results show that the algorithm fails to obtain feasible solutions for some instances with large number of customers and tight time windows. In [14] Calvo uses the solution of an assignment problem to construct feasible sub-routes, which then fuses on a main route. Calvo's algorithm [14] works better than Gendreau's [12], for instances proposed in [14], but nevertheless, feasible solutions are not found for some of them.

The objective function used in this process is the sum of all positive differences between the time it takes to reach each customer and end time, i.e. $\sum_{i=1}^{n} max(0, v_i - b_i.)$ The algorithm is executed iteratively until it finds a feasible solution (a solution with objective function equal to zero). The level variable is used to control the extent of the perturbation process and is initialized to the value 1. Then, the algorithm generates a random or sorted, possibly non-feasible solution. Afterwards, a local search process is applied using the neighborhood of a node shift (1-shift neighborhood), wherein a neighboring solution is obtained by repositioning a given client to the original solution.

The levelMax parameter is used to control the maximum level of perturbation. The algorithm is repeated until a feasible solution is obtained or until the maximum level of perturbation (levelMax) is reached. The solution is perturbed, making random 1-shift movements, at the defined level in order to escape from the current local optimal solution. Then a local search procedure, based on 1-shift procedure, is applied to the disordered solution. The solution obtained after the local search is set as the new best solution, if it is better than the current best solution. If not we keep the current best solution and we continue our local search procedure. At the end of each iteration, the value of the level variable increases by one until the current solution has not improved or the level variable has reached the levelMax. Otherwise, if there's improvement, then the level value is set to 1.

V. Our implementation

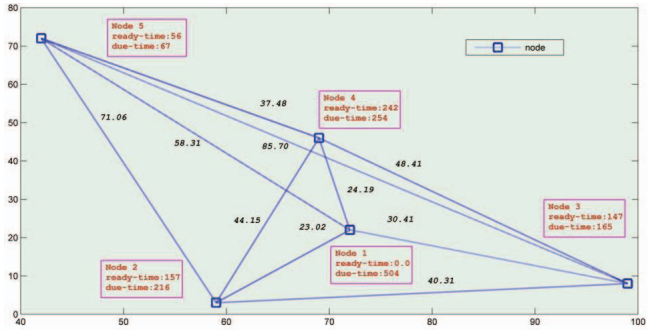

Fig. 1. A simple graphical example of TSP-TW with 5 nodes.

We implemented a GVNS process for the TSP-TW based on the description of the procedure proposed in [9]. For the particular implementation we also followed [4]. In this paragraph will be discussed the design and implementation process. The description of the process that follows is based on the method for solving the traveling salesman problem with time windows described in [9].

Procedure: VNS for finding initial solution
Input: an unfeasible solution X
Output: a feasible solution X

```
1   Initialization of feasibility distance matrix
2   Do
3       X < Construct an initial solution()
4       level < 1
5       X < LocalSearch(X)
6       Until solution is not feasible and level <= 8
7           X'< Petrubation(X,level)
8           X''< LocalSearch(X')
9           If X'' is better than X then
10              X < X''
11              level < 1
            Else
12              level < level+1
            End if
        End Until
    While the solution is not feasible
```

Fig. 2. Pseudocode of the Meta-heuristic.

This procedure appearing on Fig. 2 is the main one and concerns the idea of finding a feasible solution with the use of GVNS. Firstly, in line 1 we construct a table that shows (before our main algorithm starts) which other nodes are accessible from each node and which of them are not, according to the limitation of time window, and specifically according to the visited node's due time value. The main algorithm uses this process to exclude from checking all possible routes which do not meet the time constraints. Below is an illustrative example.

In this example we have 2 nodes; Node2 is not accessible from Node1 because the time window has expired, whereas Node1 is still accessible from Node2. As we see in the example, the first node has start time 10 and the second node end time 5, therefore it is impossible to accept a visit from the

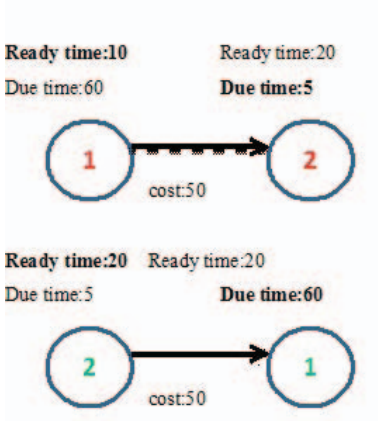

Fig. 3. Due time example-Accessibility condition.

first node, since the window will expire. However, the reverse process is allowed, so the feasible access table would be the Table I.

TABLE I. FEASIBLE ACCESS TABLE.

	Node 1	Node 2
Node 1		Not Accessible
Node 2	Accessible	

The process from line 2 to the end is repeated until a feasible solution is found. In the third line a random or sorted solution is generated. In the fourth line we set the level equal to 1. The level is used to determine the number of repetitions of the shake function, that will allow us to escape from current local minimal and accept the solution generated. The maximum level is set to 8, since this has been found to be the best from the experiments of Da Silva et al. [9]. Initially, before entering the loop, we perform a first local search hoping that the problem can be solved directly. The loop of the main algorithm runs while the variable level is less than 8 and while a feasible solution has not been found. Inside the loop a disordered solution is produced based on the value of level. Then a local search is performed and if the produced solution (after the local search) is better than the current, then we store it as the current best solution. If a new best solution is found, then we set the level variable equal to 1. If the solution after the local search is not better than the current best, then we increase the value of level parameter and continue to the next iteration. It should be mentioned that the best solution is the one with the fewest violated nodes. If two solutions have the same number of violated nodes, then we check for the cost and keep the solution with best one. Let us note that a violated node is a node that has been visited after its expiration time. The individual processes of GVNS are explained hereinafter.

The main contribution of this work lies in the process of selecting an initial solution (line 3 in Fig. 2). We study two types of initial solutions; a random type and a sorted type. Both types of initial solutions are constructed from benchmark problems. In the random initial solution, the nodes are permuted randomly. In the sorted initial solution, the visiting order of the nodes is based on the expiration time of the nodes in ascending order.

A. Local Search, Perturbation Function

The local search uses a 1-shift procedure. This procedure tries to improve the current solution by moving some nodes in the series along with some others. It is easy to conclude that if the current solution is not feasible, then two sets arise: the violated nodes (nodes that were visited after their due time) and the non violated nodes (nodes that were visited before their due time). Thus, we can separate the main neighborhood in two new sub-neighborhoods; in the first neighborhood the movements consist of violated nodes and in the second the not violated nodes participate in movements procedure.

Since we have two neighborhoods, it is important to define the ordering in which we will explore them. According to Da Silva et. al. [9] who made a thorough investigation of this case, the best order of neighborhood investigation is the following:

1) Move backwards violated nodes
2) Move forward not violated nodes
3) Move backwards not violated nodes
4) Move forward violated nodes

It should be noted that this order performs better than other rows, where the term “best” can be seen as one combination of neighborhoods that produces faster a feasible solution. The process of perturbation solution is used as entrapment avoidance procedure -shaking function- from the local minimum. In our implementation, our algorithm performs shifts (1-shift) on nodes randomly, to some randomly selected nodes in the current solution, without taking into account any control condition to shift to a better solution.

B. Implementation Technology and Instances Used

Our implementation was coded in JAVA. An important reason was the benefits of using an object-oriented language to such an optimization problem implementation.

We tested our algorithm with various benchmark problems proposed by [14] and [15], containing symmetric instances. The name of each benchmark problem has the following structure; the letter n (,i.e., number of customers), followed by the number of customers. Then, there is the letter w (,i.e., maximum window width) followed by the value of the maximum window length. For example the file name n20w100 concerns a problem with 20 nodes clients and a maximum window length of 100.

VI. COMPUTATIONAL RESULTS

TABLE III. ABBREVIATIONS FOR THE TABLE II

Abbreviation	Full Name
N	Number of nodes
Max WL	Max Window Length
Best Sol	Best Solution
RSC	Random Solution Cost
RST	Random Solution Time
dif. from best sol.	Difference from best solution
SSC	Sorted Solution Cost
SST	Sorted Solution Time

A. Random solution vs Sorted solution

The contribution of this paper is the extensive comparative study of Random vs. Sorted initial solution. The computational

TABLE II. COMPUTATIONAL RESULTS-RANDOM/SORTED FUNCTION COMPARATIVE STUDY.

Name	N	Max WL	Best Sol.	Proposed by	RSC	RST	dif. from best sol.	SSC	SST	dif. from best sol.
n20w120	20	120	265.6	Ohlmann and Thomas (2007)	357.49	0.242	34%	371.78	0.081	39%
n20w140	20	140	232.8	Ohlmann and Thomas (2007)	337.05	0.222	44%	370.61	0.049	59%
n20w160	20	160	218.2	Ohlmann and Thomas (2007)	331.00	0.163	51%	398.08	0.031	82%
n20w180	20	180	236.6	Ohlmann and Thomas (2007)	336.58	0.413	42%	418.13	0.029	76%
n20w200	20	200	241.0	Ohlmann and Thomas (2007)	365.87	0.202	51%	414.70	0.052	72%
n40w120	40	120	360.0	Calvo (2000)	521.18	12.485	44%	549.97	0.265	52%
n40w140	40	140	348.4	Calvo (2000)	493.35	41.671	41%	563.83	0.320	61%
n40w160	40	160	326.8	Ohlmann and Thomas (2007)	482.20	146.295	47%	553.81	7.190	69%
n40w180	40	180	326.8	Calvo (2000)	532.49	121.664	62%	561.09	2.240	71%
n40w200	40	200	313.8	Ohlmann and Thomas (2007)	493.65	19.919	57%	558.65	1.163	78%

results of this comparative study are listed in Table II. Our tests revealed some issues regarding the sorting approach. Specifically, they showed that for some benchmarks it is not working properly, and in general, it does not work for all ranges of time windows. We constructed some new benchmark problems and we performed some experiments on our main algorithm in order to test this hypothesis. More specifically, we generated three different data sets (benchmark problems). In the first category, all nodes were set with due time equal to zero, then with due time equal to greater ready time plus one, and at last, with the same ready time and due time.

Our experiments were conducted in a data set of well known benchmark problems that have already been proposed in the literature and also in some benchmark problems we generated for this study, in order to cover all possible cases and also, to find out in what cases this heuristic cannot properly terminate and return an initial solution for the TSP-TW. Note that the structure of the new generated benchmark problem is the same as the structure of proposed benchmarks from the literature. Also, the conditions and restrictions that we apply on the new benchmarks contain some instances that were not included in other benchmark sets.

From our experiments we can conclude that when due time was set equal to 0 for all the nodes, our algorithm did not work neither with random nor sorted initial start function. This is something that we expected in order to continue and proceed our experiments on the other, new generated benchmark categories. Furthermore, when the benchmark problem has due time equal to the largest ready time value for all nodes, then our implementation is working properly either with sorted or random as the initialization function. Finally, when on our benchmark problems which set their due time equal to largest ready time +1, for all nodes, our extended experiments show that our implementation is working properly when we initialized with a random solution, whereas it never runs with success, with sorted initialization solution.

We found that for some cases, this heuristic cannot provide feasible solution if a sorted function is used for initialization. Some previous studies mentioned that this kind of heuristic can provide feasible solutions for all proposed benchmark problems either with sorted or random function as an initialization function. They argue that random function "helps" the main algorithm to produce feasible solutions in shorter time, contrary to the choice of a sorted function. According to our research results, we show that a sorted initial function is not even working for some data sets, respecting the definition and limitations of TSP-TW.

VII. FUTURE WORK AND CONCLUSION

In the context of this paper we developed an implementation of GVNS heuristic for solving the TSP-TW in the programming language JAVA. Our approach concentrates on finding initial feasible solutions. The optimization of the initial feasible solution by using new neighborhood structures can be investigated. Furthermore, the search for alternative random procedures with different probability distributions for this algorithm could draw the attention for future studies.

Our main contribution consists of:

1) A comparative study between random and sorted approach is used to reveal an initial solution.
2) A proof-of-concept of examples which demonstrating that for many instances we are unable to find an initial solution by using the sorted approach.

The TSP applications and its variants span over several cognitive areas, including mathematics, computer science, operational research, engineering and electronics. There are three aspects in the history of any mathematical problem: how it emerged, how the research affected other mathematical developments and how it was (or not) resolved. In fact, the TSP is the most prominent of the unsolved problems of combinatorial optimization.

The TSP-TW is similar to the TSP except that cities (or customers) should be visited within a certain time window. This additional time constraint limits the search tree, because all TSP solutions are not necessarily TSP-TW solutions. However, this makes the problem even more difficult in practice. Indeed, the elegant symmetry of TSP, where any permutation of cities is a feasible solution, violated and even the search for finding feasible solutions becomes a hard problem [10]. Notice how the depot is very important for TSP-TW, while this is not for the general version of the TSP. The intractability of TSP-TW problem depends not only on the number of nodes, but also the "width" of the time window. Many efforts on exact or heuristic algorithms, that use constraint programming to solve the TSP-TW are described in literature. However related literature to this problem is rare.

REFERENCES

[1] Voigt, B.G. (1831). "Der Handlungsreisende wie er sein soll und was er zu tun hat, um Auftrge zu erhalten und eines glcklichen Erfolgs in seinen Geschften gewi zu sein von einem alten Commis-Voyageur" (The traveling salesman how he must be and what he should do in order to get commissions and be sure of the happy success in his business by an old commis-voyageur) Republished: 1981, Verlag Bernd Schramm, Kiel. [1:3]

[2] Lawler, E.L, Lenstra, J. K. Rinnooy Kan, A. H. G., Shmoys, D. B. (1985). The Traveling Salesman Problem, John Wiley and Sons.

[3] http://en.wikipedia.org/wiki/Metaheuristic(Last Accessed: 17/10/2014)

[4] Mladenovic, N., Todosijevic, R., Urosevic, D. (2012). “An efficient GVNS for solving Traveling Salesman Problem with Time Windows”, Electronic Notes in Discrete Mathematics 39 , pp.83-90.

[5] Langevin, A., Desrochers, M., Desrosiers, J., Glinas, S., Soumis, F. (1993). “A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows”, Networks 23 (7) (1993), pp. 631-640.

[6] Dumas, Y. Desrosiers, J., Gelinas, E., Solomon, M.M. (1995). “An Optimal Algorithm for the Traveling Salesman Problem with Time Windows”. Operations Research 43(2): pp.367-371.

[7] Pesant, G., Gendreau, M., Potvin, J.-Y., Rousseau, J.-M. (1998). “An exact constraint logic programming algorithm for the travelling salesman problem with time windows”, Transportation Science 32 (1998), pp.12-29.4

[8] Focacci, F., Lodi, A., Milano, M. (2002). “A hybrid exact algorithm for the TSPTW”, INFORMS Journal on Computing 14 (4) (2002), pp.403-417.

[9] Da Silva, R.F., Urrutia, S.N. (2010). “A General VNS heuristic for the traveling salesman problem with time windows”. Discrete Optimization 7(4), pp.203-211.

[10] Salvesbergh, M.W. (1985). “Local search in routing problems with time windows”, Annals of Operations Research 4 (1985), pp. 285-305.

[11] Carlton, W.B., Barnes, J.W. (1996). “Solving the travelling salesman problem with time windows using tabu search”, IEEE Transactions, 28 (1996), pp.617-629.

[12] Gendreau, M., Hertz, A., Laporte, G., Stan, M. (1998). “A generalized insertion heuristic for the travelling salesman problem with time windows”, Operation Research, 46 (1998), pp.330-335.

[13] Gendreau, M., Hertz, A., Laporte, G. (1992) “New insertion and postoptimization procedures for the travelling salesman problem”, Operations Research, 40 (1992), pp.1086-1094.

[14] Calvo, R.W. (2000). “A new heuristic for the travelling salesman problem with time windows”, Transportation Science, 34 (2000), pp.113-124.

[15] Ohlmann, J.W., Thomas, B.W. (2007). “A compressed-annealing heuristic for the travelling salesman problem with time windows”, INFORMS Journal on Computing, 19 (2007), pp. 80-90.

[16] Lopez-Ibanez, M., Blum, C. (2010). “Beam-ACO for the travelling salesman problem with time windows”, Computers and Operations Research, 37 (2010), pp.1570-1583.

[17] http://www.cleveralgorithms.com/nature-inspired/stochastic/variable_neighborhood_search.html, (Last Accessed: 17/10/2014)

[18] Yang, X.-S. (2011). “Metaheuristic Optimization”. Scholarpedia, 6 (8): 11472.

[19] Bianchi, L., Dorigo, M., Gambardella, L. M. Gutjahr, W. J. (2009). “A survey on metaheuristics for stochastic combinatorial optimization”, Natural Computing 8 (2) , pp.239-287.

[20] Blum, C. Roli, A. (2003). “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. ACM Computing Surveys (CSUR) 35, 3 (2003), pp.268-308.

[21] Glover, F. W. Kochenberger, G. A. (2003). Handbook of Metaheuristics, Springer, Hardcover.

[22] Goldberg, D.E. (1989). “Genetic Algorithms in Search, Optimization and Machine Learning (1st ed.)”. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[23] Talbi, E.G. (2009). “Metaheuristics: From Design to Implementation”. Wiley Publishing, vol 74.

[24] Mladenovic, N. Hansen, P. (1997). “Variable neighborhood search”, Computations Operations Research 24 (11) (1997), pp. 1097-1100.

[25] Mladenovic, N. (1995). “A variable neighborhood algorithm - A new metaheuristic for combinatorial optimization”, in Abstracts of papers presented at Optimization Days.

[26] Sörensen, K. (2013). “Metaheuristicsthe metaphor exposed”. International Transactions in Operational Research, 22(1), 3-18.

[27] Jarboui, B., Sifaleras, A., Rebai, A. (2015). “3rd International Conference on Variable Neighborhood Search (VNS’14)”. Electronic Notes in Discrete Mathematics, 47, 1-4.

[28] Sifaleras, A., Uroevi, D., Mladenovi, N. (2012). “EURO Mini Conference (MEC XXVIII) on Variable Neighborhood Search”. Electronic Notes in Discrete Mathematics, 39, 1-4.