20th International Conference on Modelling and Applied Simulation
18th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0037 ISBN 978-88-85741-61-4 © 2021 The Authors.
doi: 10.46354/i3m.2021.mas.001

A hybrid heuristic algorithm for solving the Traveling Salesman Problem with Time Windows

Mattia Neroni[1,*] and Letizia Tebaldi[2]

[1]Department of Engineering and Architecture – University of Parma, Parco Area delle Scienze 181/A, 43124, Parma (Italy)

[*]Corresponding author. Email address: mattia.neroni@unipr.it

Abstract

Several issues related to the logistics field can be recognized as applications of the renowned Traveling Salesman Problem with Time Windows (TSPTW); examples of these issues include, among others, instance planning deliveries, managing internal logistics, bank couriers, material handling, but also production scheduling. In the light of such numerous applications, in this paper a hybrid algorithm based on the Divide-And-Conquer (DAC) technique and the Biased Randomized heuristic Algorithm (BRA) for solving the mentioned problem is presented. The aim is to propose a flexible solution suitable for implementation in many contexts where the TSPTW is relevant, thus improving performance and key indicators. The quality and reliability of the tool are validated on several benchmark problems through a comparison with a different algorithm already proposed in literature. In the light of the simulations carried out, it turned out to be effective and efficient when dealing with problems similar to those that characterize real applications, even in terms of computational time efficiency.

Keywords: Traveling Salesman Problem; Time Windows; Divide-And-Conquer; Biased Randomized Algorithm; Logistics; Simulation.

1. Introduction

Logistics and transportation activities are strategic elements for the firm success in gaining competitive advantage (Penteado et al., 2016). The adoption of well-designed logistic procedures may lead to great benefits such as labor saving, costs reduction, mitigation of the bullwhip effect, lead time reduction, and decreased risk of stock out (Zhang and Lai, 2006). Decision makers involved in logistics and transportation have constantly to face several challenges due to the globalization, the advent of e-commerce, the continuous pressing for sustainable models, and the increased complexity of supply chains.

For instance, within a warehouse, concerns could be related to the picking process, which contributes up to 60-70% of the total warehouse costs (Kulak et al., 2012), in terms of optimizing the route the picker (or the robot nowadays) has to travel, or the best storage allocation scenario allowing the order picking to be improved (e.g. Öztürkoğlu, 2020); transports as well constitute a key question, which very often results in techniques allowing to determine the best journeys able to minimize travel time and, consequently, costs and emissions (e.g. Eshtehadi et al., 2020).

Many other issues related to the logistics field could be addressed, and what they all have in common is their ultimate aim of optimization.

In most cases, the best solution is provided by the operational research field (Dekker et al., 2012); indeed, supporting decisions within the logistics context is one of the most successful areas of simulation and optimization tools (Juan et al., 2015).

One of the most debated problems in the context of logistics is to determine the design of delivery routes for vehicles through a set of geographically scattered customers subject to side constraints. If we think that in the last years over 70% of goods were transported by road (Eurostat, 2020), the relevance of this issue immediately follows. This aspect is well-known in literature as Traveling Salesman Problem (TSP). The simplest description of the TSP can be resumed as follows: the salesman must visit a set of customers (or sale points), and, given the travel cost sustained to move from a customer to the other, he/she has to determine the cheapest tour connecting them all, visiting each customer only once, and returning to the origin point (Cheng et al., 2008), namely the depot. Clearly, the travel costs could also be interpreted as travel times or distances, and, in this case, the objective should be to make the journey as short as possible.

One of the variants of the TSP resulting from the application of the problem in real contexts, is the Traveling Salesman Problem with Time Windows (TSPTW), whose complete mathematical formulation is provided for instance by Ferreira da Silva and Urrutia (2010). It mainly consists in a TSP where the customers to be visited are subject to time constraints and must be reached within a specific time span, namely the time window.

Both these problems find application in many contexts, not limited to transportation issues only. Indeed, as Silva et al. (2020) state, algorithms used for the TSP can be used as well for solving order picking problems in manual warehouses by considering the picker as an alter ego of the salesman, or currently the drone delivery is spreading and the drone itself can be seen as a vehicle; an application example of this kind is provided by Briant et al., (2020). Moreover, it was recently implemented for job scheduling issues (e.g. Ahmadov and Helo, 2018), for time optimization analysis (e.g. Selvi et al., 2019), for developing models for lot-sizing problems under capital flow constraints (e.g. Chen and Zhang, 2018) or even in systems for intelligent water irrigation and fertigation (e.g. Karunanithy and Velusamy, 2020), for data collection (e.g. Qu et al., 2020) and many others. Hence, the versatility of both TSP and TSPTW and their multiple fields of applications make them significant and relevant.

However, while for the TSP many interesting solutions have been proposed over the years, the same cannot be said for the TSPTW. Carrying out an easy query on the Scopus database, and looking at the number of publications concerning the TSP and the TSPTW, the imbalance is clear. For the TSP there are more than 12,000 publications, *versus* the only around 170 for the variant with time windows, which nonetheless, surely deserves attention, as it was demonstrated that the most critical constraint companies have to face is exactly represented by the time windows (Bychkov and Batsyn, 2018).

In an attempt to partially fill this gap, in this paper the TSPTW is tackled through a hybrid solution which relies on the well-known Divide-And-Conquer (DAC) technique and the Biased Randomized Algorithm (BRA) (Juan et al., 2010). To the best of the authors' knowledge, there is no evidence in literature of a hybrid version built on the basis of these two approaches, despite many studies in different contexts prove that both tools are very efficient and computationally fast.

The remainder of the paper is structured as follows. Firstly, two brief digressions on the DAC and the BRA are introduced, respectively in section 2 and 3. The proposed algorithm is presented in section 4, and then, to demonstrate the quality and the reliability of the new solution, the results of several simulations are presented in section 5, where the proposed approach is compared with three others algorithms on different benchmark problems. Finally, conclusions and future perspectives are provided in section 6.

2. The Divided-And-Conquer Technique

The Divide-And-Conquer, also referred to as *divide-et-impera*, is a well-known algorithm design paradigm. It basically consists in recursively breaking down a problem into two or more sub-problems, until these become simple enough to be solved directly. The solutions of the sub-problems are then combined together to provide a unique solution to the original version of the main problem. It is clear that this approach refuses *a priori* the achievement of the global optimum; however, it brings significant benefits of paramount importance when approaching complicated combinatorial optimization problems. Indeed, first of all it is proved to be both very efficient and very performant; secondly, it is naturally adapted to be used on multi-processor machines, and tends to make an efficient use of memory caches.

The DAC technique is the basis of efficient algorithms for many problems, such as search algorithms (e.g., binary search), sorting algorithms (e.g., quicksort, merge sort), large numbers multiplications with floating points round off control (e.g., the Karatsuba algorithm), and many others. It is widely discussed by the scientific community and adopted in many engineering and mathematics problems (see for instance interesting literature reviews of its applications by Bontempi and Birattari, 2005; Mei et al., 2016; Yang et al., 2019).

According to its definition and formalization, the TSPTW is well-suited for being recursively broken down into smaller problems, and this is exactly what the DAC technique does. Indeed, its application to the traditional TSP is not new in literature: for instance, Meuth and Wunsch (2008) applied it to TSP for vehicle routing obtaining good results, while Mulder and Wunsch (2003), even if without getting satisfactory results, solved a TSP with even 1 million nodes in a very short computational time by aggregating them in small subsets using a neural network (they were probably inspired by a previous similar work by Foo and Szu, 1989). Nonetheless, to the authors' best knowledge, there are no studies that implement the DAC to the TSPTW, especially in combination with a Biased Randomized Algorithm.

3. The Biased Randomized Algorithm

The Biased Randomized Algorithm (BRA) belongs to the plethora of randomized heuristics that, nowadays, are widely used to solve large scale optimization problems. It might be classified as a constructive procedure, since the solution is iteratively built one element at a time, although it is frequently incorporated in a metaheuristic framework, such as an iterated local search (see for instance Juan et al., 2014). In line with the similar concept of the roulette wheel, in the BRA each element is selected according to a certain probability: the greater is the benefit obtained by introducing an element at that point of the construction, the greater is the probability to choose it. The idea behind this concept is to introduce slight modifications in the greedy constructive behavior, to escape the local optima by exploring many solutions in a very short computational time, while maintaining the logic behind the heuristic.

The probability mentioned few lines above might be calculated according to several different criteria, such as ranking, priority rule, heuristic value, and many else. The first BRAs were proposed by Arcus (1965) and Tonge (1965), who named it Biased Random Sampling (BRS) and used it to bias the selection of randomly generated solutions. In the following years, many priority rules-based heuristics have been designed, although the first application of a BRA in a metaheuristic framework came 24 years later, when Glover (1989) proposed his Probabilistic Tabu Search (PTS), successively extended in Glover (1990). Another metaheuristic famous for integrating a BRA is the Ant Colony Optimization (ACO), originally introduced by Colorni et al. (1991). All the above-mentioned implementations define the probability by using an empirically constructed distribution; despite that, using a theoretical distribution it is possible to obtain a random element in a less time-consuming way by using an analytical expression. In this way, Juan et al. (2010) were pioneers in the implementation of a skewed theoretical distribution in the BRA. The candidate solutions are therefore sorted from the best one to the worst one according to the desired criterion, and then the probabilities are assigned to the candidates depending on their position in the list. According to the authors' experience, the most common theoretical distribution in BRA is the quasi-geometric distribution described in equation (1). The reason for its popularity is probably that it depends on a single parameter α, which avoids time-consuming fine-tuning processes for parameters' setting (Juan et al., 2015).

$$f(x) = (1-\alpha)^x \quad (1)$$

Note that, for α very close to 1 a greedy solution is always returned, while for α very close to 0 it approximates a uniform distribution.

For further implementations and additional deepening, the authors suggest Grasas et al. (2017), who carried out a recent literature review on this specific topic.

4. The proposed algorithm

4.1. Problem formulation

The TSPTW consists in the construction of a route to visit a set of M nodes, *alias* customers $(j = 1,\dots,M)$ by minimizing the travelling distance/cost/time, under temporal constraints, i.e., the time windows. The starting and the ending points always match with the origin (e.g., in real applications the deposit, the entry point, or the logistic HUB); indeed, the problem might also be understood as the definition of a Hamiltonian Cycle. The time constraint imposes that each node j, origin included, must be visited within a specific timeframe which goes from its opening time (i.e., s_j) to its closing time (i.e., e_j). The violation of these time windows generally involves an additional cost or a penalization. A solution might be formalized as an array containing the M nodes, sorted in the order in which they are supposed to be visited. As already stated, the first and the last element of the solution must coincide with the origin.

In the proposed algorithm, the cost of a solution is intended to be the total time needed to complete the tour through all the nodes (i.e., cost-in-time), plus an additional cost due to eventual delays. It follows that the objective is to minimize this value. The cost of a solution is provided in equation (2).

$$cost = \sum_{j=2}^{M} (\tau_j + max\{0;\ \tau_j + p_j - e_j\}) \quad (2)$$

where:

- p_j is the processing or service time at node/customer j;
- e_j is the closing time of node j;
- τ_j is the time in which node j is reached and, given $d_{j-1,j}$ the distance-in-time between nodes $j-1$ and j and s_j the opening time of node j, it is calculated as $max\ \{\tau_{j-1} + p_{j-1} + d_{j-1,j};\ s_j\}$. This is true for $j \in [2, M]$, because of course $\tau_1 = 0$;
- $max\{0; \tau_j + p_j - e_j\}$ is the penalty component, which occurs in case of eventual delay.

Note that a delay can occur not only when a node is reached after its closing time, but also when it is reached on time, but the service time p_j forces it to postpone the closure, thus determining a delay.

4.2. Main procedure

The main procedure of the proposed algorithm is inspired by the classic DAC approach. The starting set of M nodes that constitute the main problem is recursively split into smaller subsets, until the number of nodes in each of them is below a certain threshold (i.e., γ). Then, each subset is solved using the incorporated algorithm (in this case the BRA), and the solutions are aggregated to constitute the final one.

The splitting process (Figure 1) is carried out as follows. If the number of nodes in the considered set is over the predefined threshold, a random node r is chosen according to a uniform probability distribution. The considered set of nodes is therefore divided into two subsets: *(i)* the first one is made of nodes that close before the opening of r, *(ii)* the other is composed of the remaining nodes.

In case after having selected the random node r the splitting is not possible, a new random node is selected. This step is repeated again and again until a splitting is obtained, or a maximum number of attempts is reached. In the latter case, the set of nodes that was not possible to divide, is optimized as-is using the BRA.

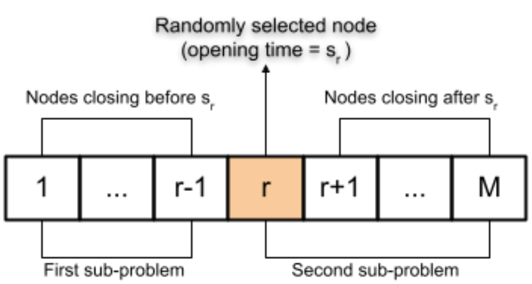

Figure 1. Representation of the problem decomposition.

4.3. The optimization of the subsets

The optimization of the subsets of nodes is made using a metaheuristic framework that makes use of the BRA. Before describing this procedure, two variables, namely α and β, should be introduced:

- $\alpha \in (0,1)$ is the parameter of the quasi-geometric distribution in equation (1),
- $\beta \in (0,1]$ is a variable that represents, in the current iteration, how much of the solution is destructed and reconstructed to create a new solution for the subset.

The procedure for the optimization of the subsets always starts with setting β at a low starting value (e.g., 0.1), and the current solution at the greedy one, in which, given node j, the next node (i.e., $j+1$) is selected as the node that minimizes the cost function in equation (3).

$$cost_{j,j+1} = max\,\{\tau_j + p_j + d_{j,j+1};\ s_{j+1}\} + max\,\{0;\ \tau_{j+1} + p_{j+1} - e_{j+1}\}. \qquad (3)$$

Then, at each iteration of the algorithm until the stopping criteria are met, given m the length of the current solution for the subset, the last $m \cdot \beta$ nodes are removed and reinserted to create a new possible solution (Figure 2). If the new solution is better than the current one, this latter is replaced and β is reset at the low starting value, otherwise β is increased in order to destruct and reconstruct a greater part of the current solution during the next iteration.

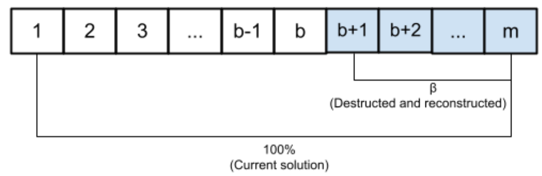

Figure 2. Representation of destruction and reconstruction process used to create new solutions.

The reconstruction of the current solution (or part of it) is made using the BRA. The nodes to append to the solution are sorted from the best one to the worst one according to equation (3) (where j in this case is the last node of the solution under construction). Each of them is assigned a probability of being included, that depends on its position in list determined using equation (1) (Figure 3); the node to include is therefore randomly selected. The process is repeated until the new solution is complete.

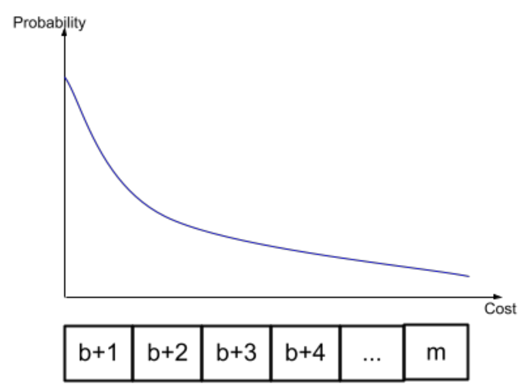

Figure 3. Representation of the selection of each node included in the solution.

5. Validation and results

The algorithm was implemented in Go© programming language and tested on a standard personal computer Intel Quad Core i7 CPU at 3.6GHz with 8Gb RAM and Ubuntu 18.04© operative system. Being Go garbage collected, the program does not execute as fast as those written in C or C++;

nonetheless, it turns out to be reasonably fast for a real implementation, as also demonstrated by the computational times observed. The code is also available open-source at the following link: https://github.com/mattianeroni/Divide-Et-Impera.

In order to provide a robust validation, the algorithm was compared to that proposed by Ferreira da Silva and Urrutia (2010), which, compared to the plethora of existing algorithms for solving the TSPTW, is relatively new, and, according to the Scopus database is one of the most cited documents. Moreover, the authors of this algorithm have taken into account very complicated and big sized problem, and, as proven in their paper, they already outperform two old but very important algorithms, such as the generalized heuristic by Gendreau et al. (1998), and the simulated annealing with variable penalty described in Ohlmann and Thomas (2007).

Before carrying out a comparison, a parameters tuning is needed. In this respect, the proposed algorithm offers an additional advantage. As matter of fact, it has very few parameters to optimize, and, as shown by the parameters tuning below, it is quite insensitive to them. The parameter of the quasi-geometric distribution α has been set equal to 0.9 according to the suggestions found in literature (Grasas et al. 2017). The only two remaining parameters are *(i)* the predefined length of the subsets (i.e., γ), and *(ii)* the number of iterations for the BRA. Four possible combinations of these parameters have been tested on three problems of different complexity, chosen from the benchmarks successively used for testing. The selected values for γ are respectively 30 and 50, while the tested number of iterations for the BRA are 1500 and 3000. We are aware that the greater is the number of iterations the higher is the possibility to have a better solution; however, at first, we believe a trade-off between performance quality of the solution is due, secondly, this is true only into the single subsets of node and not for the final complete solution.

The results of the parameters tuning are presented below, in Table 1. Being the proposed algorithm subject to stochasticity, it has been iterated 10 times for each combination parameters-benchmark, and in the table are presented the average results and the standard deviations.

Table 1. Results of the parameters' tuning.

Benchmark	γ	Iterations of BRA	Avg. Cost	St.Dev. Cost	Avg. Comp. time [s]	St.Dev. Comp. time [s]
n200w100_001	30	3000	10248	61	0.921	0.098
	30	1500	10248	61	0.401	0.036
	50	3000	10213	0	1.183	0.067
	50	1500	10213	0	0.566	0.057
n400w500_005	30	3000	22114	0	2.223	0.194
	30	1500	22193	69	1.155	1.495
	50	3000	22154	69	2.659	0.323
	50	1500	22193	69	2.400	0.136
n350w200_005	30	3000	18268	0	1.571	0.080
	30	1500	18216	0	1.127	1.600
	50	3000	18320	90	2.090	0.064
	50	1500	18268	90	1.750	0.048

Results of parameters tuning show that there is no significant correlation between the parameters' value and the results of the algorithm. As expected, iterating more times the BRA, the computational time is slightly longer and the average cost is slightly lower; however, we do not consider both differences as relevant for preferring a setting instead of another. To carry out the tests, we opted for γ=30, and iterations=3000. We are aware that these parameters should be tuned again when the algorithm is implemented on problems of different average complexity, but we are confident their impact on results is not crucial.

The results of testing and simulations are presented in Appendix A at the end of the manuscript (Table 2), where the proposed algorithm is compared in terms of cost of the best solution and computational times with results from the algorithm proposed by Ferreira da Silva and Urrutia (2010). Again, since our algorithm is subject to stochasticity, it has been iterated 10 times on each benchmark, and the results presented in Table 2 refer to the average result and the standard deviation calculated on these 10 iterations.

For the implementation of the algorithm proposed by Ferreira da Silva and Urrutia, we used the C++ implementation open-sourced by the authors at the following link: https://homepages.dcc.ufmg.br/~rfsilva/tsptw/#instances.

The benchmark values have been taken from the same repository, and their nomenclature can be interpreted as follows. Given the name of a benchmark problem (say for instance n200w100_001, the first benchmark of Table 2), the number after the 'n' represents the number of nodes, the number after 'w' represents the size of the time windows, and the last three numbers are a unique ID to distinguish that problem from others problems with the same characteristics.

As presented in Appendix A, the proposed algorithm is always able to outperform the one proposed by Ferreira da Silva and Urrutia. On average the proposed solutions are 0.85% better, and the algorithm is extremely reliable, since the coefficient of variation (σ/μ) calculated on the presented results is always less than 1%. The computational time is surprising. The proposed algorithm is 100÷1000 times faster, even if the comparison algorithm was implemented in C++. A comparison of the number of solutions explored would allow us to go deeper into this difference, although we have not been able to do it because our algorithm is iterating more times on subsets of nodes only. A gross estimate made on the basis of the average number of subsets in which each problem is split says that our algorithm explores less solutions, and this might be the reason for the shorter computational time.

6. Conclusions

This paper aimed at presenting a hybrid algorithm developed on the bases of the Divide-and-Conquer approach and the Biased Randomized Algorithm for solving the Traveling Salesman Problem with Time Windows, a common problem implemented to solve logistics issues. The solution has been designed for planning transportation activities; although, it can be implemented in several other contexts where the TSPTW can find application. The proposed algorithm has been compared to another algorithm proposed by the scientific community, and it turned out to be very efficient (seeking a better solution in all the analyzed benchmarks) and obtaining it in surprising short computational times.

It presents of course some limitations and it lacks realism in the assumption of cost-in-time. We therefore aim to better explore these criticalities in occasion of future works. More in detail, the possible future research perspectives may concern: *(i)* the testing and implementation of the same algorithm in some real contexts of application of the TSPTW such as transportation or production scheduling, essential step to refine the algorithm since it would let emerge practical issues and concerns which can be observed only after real implementations; *(ii)* the consideration of customer-dependent delay penalties, in a scenario where there are some trusted and prominent customers, and a delay in delivery to these customers would have a greater impact; *(iii)* the combination of the proposed algorithm with the well-known Clarke-Wright savings algorithm, in order to apply it to a Vehicle Routing Problem with Time Windows (El-Sherbeny, 2010).

Moreover, once the algorithm will be adapted to deal with more vehicles, a real case study of a company operating within the field of express deliveries is intended to be carried out: firstly, historical data as far as the travel time of their journeys will be recorded; then, the algorithm will be operatively implemented for a sufficient time in order to assess whether this solution could lead to tangible benefits and savings

References

Ahmadov, Y. and Helo, P. (2018). A cloud based job sequencing with sequence-dependent setup for sheet metal manufacturing. *Ann. Oper. Res.,* 270:5-24.

Arcus, A. L. (1965). A computer method of sequencing operations for assembly lines. *Int. J. Prod. Res.,* 4:259-277.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L. and Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *Eur. J. Oper. Res.*, 285:497-512.

Bontempi, G., & Birattari, M. (2005). From linearization to lazy learning: A survey of divide-and-conquer techniques for nonlinear control. *International Journal of Computational Cognition*, 3(1).

Bychkov, I. and Batsyn, M. (2018). A Hybrid Approach for the Capacitated Vehicle Routing Problem with Time Windows. *Proceedings of the School-Seminar on Optimization Problems and their Applications (OPTA-SCL 2018).*

Chen, Z. and Zhang, R. (2018). A capital flow-constrained lot-sizing problem with trade credit. *Sci. Iran.*, 25:2775-2787.

Cheng, W., Maimai, Z. and Jian, L. (2008). Solving traveling salesman problems with time windows by genetic particle swarm optimization. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*,1752-1755.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991). Distributed optimization by ant colonies. *In Proceedings of the first European conference on artificial life*, 142:134-142.

Dekker, R., Bloemhof, J. and Mallidis, I. (2012). Operations Research for green logistics – An overview of aspects, issues, contributions and challenges. *Eur. J. Oper. Res.*,, 219:671-679.

El-Sherbeny. (2010). Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *J. King Saud Univ. Sci.*, 22:123-131.

Eshtehadi, R., Demir, E. and Huang, Y. (2020). Solving the vehicle routing problem with multi-compartment vehicles for city logistics. *Comput. Oper. Res.*, 115, A.N. 104859.

Eurostat, (2020). Statistics Explained, Freight Transportation Statistics – modal split. https://ec.europa.eu/eurostat/statistics-explained/pdfscache/1142.pdf [Accessed January 2021].

Ferreira da Silva, R. and Urrutia, S. (2010). A General VNS heuristic for the traveling salesman problem with time windows. *Discrete Optim.*, 7:203-211.

Foo, Y. S., & Szu, H. (1989). 'Solving large-scale optimization problems by divide-and-conquer neural networks'. IEEE International Joint Conference on Neural Networks, Vol. 1, pp. 507-511.

Gendreau, M., Hertz, A., Laporte, G. and Stan, M. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows. *Oper. Res.*, 46:330-335.

Glover, F. (1989). Tabu search—part I. *ORSA J. Comput.*, 1:190-206.

Glover, F. (1990). Tabu search—part II. *ORSA J. Comput.*, 2:4-32.

Grasas, A., Juan, A. A., Faulin, J., de Armas, J. and

Ramalhinho, H. (2017). Biased randomization of heuristics using skewed probability distributions: a survey and some applications. *Comput. Ind. Eng.*, 110:216-228.

Juan, A. A., Faulin, J., Ruiz, R., Barrios, B. and Caballé, S. (2010). The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. *Appl. Soft Comput.*, 10: 215-224.

Juan, A. A., Lourenço, H. R., Mateo, M., Luo, R., & Castella, Q. (2014). Using iterated local search for solving the flow-shop problem: parallelization, parametrization, and randomization issues. International Transactions in Operational Research, 21(1), 103-126.

Juan, A. A., Pascual, I., Guimarans, D. and Barrios, B. (2015). Combining biased randomization with iterated local search for solving the multi-depot vehicle routing problem. *Int. T. Oper. Res.*, 22:647-667.

Karunanithy, K. and Velusamy, B. (2020). Energy efficient cluster and travelling salesman problem based data collection using WSNs for Intelligent water irrigation and fertigation. *Measurement: Journal of the International Measurement Confederation*, 161:107835.

Kulak, O., Sahin, Y. and Egement Taner, M. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flex. Serv. Manuf. J.*, 24:52-80.

Mei, Y., Omidvar, M. N., Li, X., & Yao, X. (2016). A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. ACM Transactions on Mathematical Software (TOMS), 42(2), 1-24.

Meuth, R. J., & Wunsch, D. C. (2008). Divide and conquer evolutionary TSP solution for vehicle path planning, IEEE Congress on Evolutionary Computation, pp. 676-681.

Mulder, S. A., & Wunsch D. C. (2003). Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. Neural Networks, 16(5-6): 827-832.

Ohlmann, J.W. and Thomas, B.W. (2007). A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS J. Comput.*, 19(1).

Öztürkoğlu, Ö. (2020). A bi-objective mathematical model for product allocation in block stacking warehouses. *Int. T. Oper. Res.,* 27:2184-2210.

Penteado M., M. and Chicarelli A., R. (2016). Logistics activities in supply chain business process. *Int. J. Logist. Manag.*, 27:6-30.

Qu, F., Liu, J., Ma, Y., Zang, D. and Fu, M. (2020). A novel wind turbine data imputation method with multiple optimizations based on GANs. *Mech. Syst. Signal Pr.*, 139:106610.

Selvi, L., Joelianto, E. and Leksono, E. (2019). Time Optimization Analysis Using Hybrid Simulated Annealing and Genetics Algorithm for CNC Punching Machine. *2nd International Conference on Mechanical, Electronics, Computer, and Industrial Technology, MECnIT 2018*, 1230.

Silva, A., Coelho, L.C., Darvish, M. and Renaud, J. (2020). Integrating storage location and order picking problems in warehouse planning. *Transp. Res. Part E,* 140:102003.

Tonge, F. M. (1965). Assembly line balancing using probabilistic combinations of heuristics. *Manag. Sci.*, 11:727-735.

Yang, P., Tang, K., & Yao, X. (2019). A parallel divide-and-conquer-based evolutionary algorithm for large-scale optimization. IEEE Access, 7, 163105-163118.

Zhang, G. Q. and Lai, K. K. (2006). Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. *Eur. J. Oper. Res.*, 169:413-425.

Appendix A

Table 2. Results of the numerical validation.

Benchmark	Ferreira Da Silva and Urrutia (2010)		Proposed algorithm				Benchmark	Ferreira Da Silva and Urrutia (2010)		Proposed algorithm			
	Cost	Comp. time [s]	Avg. Cost	St.Dev. Cost	Avg. Comp. time [s]	St.Dev. Comp. time [s]		Cost	Comp. time [s]	Avg. Cost	St.Dev. Cost	Avg. Comp. time [s]	St.Dev. Comp. time [s]
n200w100_001	10402	5.515	10213	0	0.566	0.057	**n350w100_005**	19238	46.337	19121	26	1.679	0.190
n200w100_002	10707	4.497	10580	28	1.049	0.111	**n350w200_001**	18199	49.523	18059	4	1.645	0.254
n200w100_003	10313	4.850	10239	1	1.025	0.100	**n350w200_002**	19190	45.863	18987	0	1.711	0.177
n200w100_004	10562	5.015	10513	0	1.124	0.413	**n350w200_003**	17594	49.235	17466	8	1.568	0.188
n200w100_005	10972	5.082	10904	20	0.909	0.232	**n350w200_004**	18539	47.076	18352	28	1.945	0.548
n200w200_001	10906	6.021	10863	0	1.174	0.515	**n350w200_005**	18421	50.149	18372	0	1.571	0.080
n200w200_002	11221	5.932	11117	4	0.981	0.038	**n350w300_001**	18603	53.818	18517	0	4.188	3.933
n200w200_003	10474	5.899	10339	21	0.904	0.080	**n350w300_002**	18453	56.712	18321	42	1.964	0.672
n200w200_004	10513	6.095	10464	0	2.133	1.937	**n350w300_003**	18386	54.103	18215	23	1.672	0.131
n200w200_005	10490	6.271	10311	0	0.931	0.187	**n350w300_004**	18071	57.652	17881	31	2.537	1.912
n200w300_001	10240	7.462	10065	8	0.853	0.095	**n350w300_005**	18489	57.806	18359	23	1.574	0.085
n200w300_002	10482	7.281	10397	50	0.882	0.040	**n350w400_001**	17551	62.707	17439	0	1.577	0.116
n200w300_003	10946	7.236	10764	40	0.831	0.127	**n350w400_002**	18318	57.423	18074	14	1.585	0.269
n200w300_004	10671	7.190	10529	14	0.838	0.032	**n350w400_003**	18302	59.344	18062	13	1.491	0.409
n200w300_005	10420	7.328	10369	0	0.783	0.078	**n350w400_004**	19420	62.279	19361	31	1.598	0.222
n200w400_001	10524	8.643	10454	67	0.981	0.208	**n350w400_005**	18249	56.102	18126	16	1.607	0.097
n200w400_002	10250	8.307	10078	48	1.157	0.393	**n350w500_001**	18918	58.864	18779	0	1.656	0.224
n200w400_003	10909	9.325	10870	0	1.872	1.093	**n350w500_002**	18499	58.356	18417	8	1.656	0.276
n200w400_004	10242	8.672	10106	31	2.403	2.690	**n350w500_003**	18789	59.703	18612	74	1.681	0.216
n200w400_005	10546	9.290	10472	44	1.026	0.433	**n350w500_004**	19635	59.197	19546	43	2.458	1.147
n200w500_001	10901	9.678	10768	80	0.795	0.076	**n350w500_005**	19379	57.516	19230	84	1.802	0.409
n200w500_002	10260	10.334	10148	20	1.054	0.203	**n400w100_001**	20089	57.246	20002	27	2.351	0.923
n200w500_003	10499	9.458	10442	17	0.933	0.192	**n400w100_002**	21056	56.473	20845	0	1.751	0.315
n200w500_004	10080	9.985	10074	0	1.024	0.174	**n400w100_003**	21334	57.498	21284	0	1.613	0.093
n200w500_005	10476	10.542	10320	154	1.045	0.409	**n400w100_004**	20975	56.028	20823	54	1.749	0.136
n250w200_001	12876	11.936	12717	47	2.284	0.724	**n400w100_005**	20395	55.923	20214	0	1.827	0.234
n250w200_002	13098	12.572	12928	33	1.405	0.414	**n400w200_001**	21260	70.165	21132	22	1.817	0.064
n250w200_003	13663	11.639	13520	2	1.458	0.590	**n400w200_002**	21604	62.211	21472	7	1.720	0.068
n250w200_004	12976	11.112	12868	99	0.954	0.146	**n400w200_003**	20769	69.053	20624	0	1.766	0.287
n250w200_005	12749	11.438	12633	0	1.537	0.409	**n400w200_004**	22169	68.588	22041	13	1.848	0.168
n250w300_001	13965	13.866	13874	0	1.179	0.110	**n400w200_005**	21815	66.221	21652	58	1.906	0.166
n250w300_002	13056	14.561	13008	0	2.567	2.159	**n400w300_001**	21779	80.853	21530	18	1.805	0.093
n250w300_003	13884	15.080	13743	93	1.041	0.069	**n400w300_002**	20102	79.865	20022	72	1.807	0.174
n250w300_004	13682	15.219	13542	2	1.698	0.685	**n400w300_003**	21367	102.188	21259	64	1.897	0.116
n250w300_005	13190	13.440	13029	68	1.498	0.564	**n400w300_004**	22926	100.945	22859	0	2.402	0.795
n250w400_001	13778	18.919	13702	58	1.209	0.332	**n400w300_005**	20655	90.303	20546	0	2.371	0.219
n250w400_002	13208	17.662	13038	15	2.314	1.142	**n400w400_001**	21125	96.707	21024	73	1.735	0.172
n250w400_003	13395	19.354	13251	62	1.340	0.030	**n400w400_002**	20857	92.308	20704	68	2.066	0.008
n250w400_004	13225	17.955	12998	0	1.365	0.261	**n400w400_003**	21579	101.614	21436	59	1.725	0.264
n250w400_005	12712	19.459	12579	0	1.860	1.066	**n400w400_004**	20198	105.519	20018	58	1.961	0.196
n250w500_001	13098	20.128	13034	0	1.432	0.612	**n400w400_005**	21654	89.905	21540	31	1.873	0.059
n250w500_002	13686	20.600	13571	13	1.047	0.123	**n400w500_001**	20073	109.416	19930	29	2.531	0.965
n250w500_003	12833	22.249	12650	89	1.319	0.216	**n400w500_002**	20965	104.076	20844	17	2.736	0.772
n250w500_004	12604	21.261	12544	18	1.347	0.249	**n400w500_003**	21551	109.525	21443	0	2.107	0.196
n250w500_005	14064	21.128	13841	54	3.455	2.905	**n400w500_004**	20506	117.584	20296	13	2.165	0.528
n350w100_003	18726	46.062	18655	18	1.686	0.374	**n400w500_005**	22329	102.001	22114	0	2.223	0.194
n350w100_004	18307	40.320	18204	42	1.520	0.196							