Journal of Engineering and Applied Sciences 13 (14): 5911-5925, 2018
ISSN: 1816-949X

The Shifting Traveling Salesman Problem: From Modeling to Resolution

[1]Amina El Yaagoubi, [2]Ahmed El Hilali Alaoui and [1]Jaouad Boukachour
[1, 2]Modeling and Scientific Computing Laboratory, Faculty of Sciences and Technologies,
Sidi Mohamed Ben Abdellah University, Fez, Morocco
[2]University Le Havre Normandie, 76600 Le Havre, France

Abstract: This study introduces the Shifting Traveling Salesman Problem (ShTSP) which is a new variant of the transportation problems that combines the well-known traveling salesman problem and the shifting problem. The ShTSP arises naturally in the transportation of large, heavy, hazardous or fragile products in a single vehicle with a single stack where all products are stowed in a predefined order according to their weight, fragility and stability. The stack has a single access point for the unloading of freight which means, the unloading of each product is performed according to the "Last In First Out" (LIFO) policy such that a number of products must be removed in order to reach products below them. In other words, shifting products within the vehicle becomes necessary if the target product is located below other ones. Our goal is to seek an optimal tour that takes account of the shifting cost which represents the temporary removal of frights in the vehicle caused by the unloading and reloading operations at each client of the tour. We propose a mathematical model as a mixed nonlinear program and then we solve it by proposing two methods: the first one consists on adapting the ant colony metaheuristic and the second one introduces a new parallel-ant colony adaptation, the two algorithms are tested on a number of problem instances of varying problem characteristics from the TSPLIB benchmark sets. Computational results show the efficiency of the improved version of the algorithm which is based on the parallel concept, for small and large sized instances.

Key words: Traveling salesman problem, unloading, reloading, shiftings, mathematical programming, optimization, metaheuristic, parallel-ant colony algorithm

INTRODUCTION

Because of their wide applicability in practical settings, routing problems have been intensively studied in the literature and known as the classical combinatorial optimization problems that have become a key component of distribution and logistics management. However, nowadays, service and transportation companies compete not only on their logistics costs but also on service differentiation that may be influenced by the shifting constraints which are practical aspects that have seldom been addressed in the modeling approaches of the literature.

Avoiding load rearrangements and temporary removal of freights is very important in the case of transporting large, heavy, fragile items or hazardous materials. In fact, we can face real-world applications problem where, both the shifting and the traveling salesman problems are combined together. These problems are encountered in several transportation companies which deliver large items such as furniture and appliances. Thus, the aim of this study is to study a new variant of the routing problems which is the Shifting Traveling Salesman Problem (ShTSP). This problem arises naturally in the routing of a single vehicle or truck that has a single access point for the unloading of freight. The vehicle can also, be represented as a container ship or a barge (Yaagoubi *et al*., 2016) where each stack of the ship/barge can be resolved as the ShTSP. The following subsections explain both problems: the Shifting Problem (ShP) and the Traveling Salesman Problem (TSP).

Overview of the shifting problem: The ShP is encountered in most freight transport problems and especially in the maritime transport problem, so, to introduce this problem we will start first by recalling the concept of containerization which is a system that involves the transportation of freight using intermodal containers. The introduction of this system has contributed a significant reduction in the cost of freight transportation. Ever since, the container has become a pillar of commerce throughout the world. Thus, the

Corresponding Author: Amina El Yaagoubi, Modeling and Scientific Computing Laboratory,
Faculty of Sciences and Technologies, Sidi Mohamed Ben Abdellah University, Fez, Morocco

container development has risen regularly and continuously and developed stunningly these recent decades. As a result of this extremely large growth, terminal operators have started to think more about efficiency and costs reductions. Thereby, a lot of multiple studies are prepared to search ways to ameliorate efficiency and save costs.

Containerization concerns the intermodal freight transportation in containers using several modes of transportation such as ships, trucks, trains and barges without any manipulation of the freight itself when changing modes (Crainic and Kim, 2007). That is, the main reason for its highly importance is that it facilitates smooth movement of goods through multiple transportation modes without any direct goods handling all along the trajectory. The containerization also plays a major role in reducing freight handling and thus improving security, decreasing damages and losses and allows freight to be transported faster (Agerschou, 2004).

Containers are large metal boxes of multiple standard dimensions (20 ft Equivalent Unit (1 TEU), 40 ft (2 TEU), 45 ft (high-cube)), allowing several units of cargo to be manipulated simultaneously and efficiently and they are constructed, so that, they can be stored efficiently and directly on top of each other in stacks. Containers are almost always stored in this way, both in stationary storage areas such as depots, warehouses and port terminal yards and moving storage areas such as bays and stacks of container ships or vessels.

Within this context, the position of containers is the most influencing factor on the cost needed to achieve certain retrieval. Sometimes because of this predefined order of the containers in the stack, the container that is due to be lifted out of the stack earlier may be buried beneath other containers that would be lifted at a later time. In order to retrieve a certain container x, all containers on its top have to be removed first. These extra movements are called shiftings and sometimes multiple shifts may be necessary to reach a certain container. Accordingly and because containerships are highly capitalized and their handling costs are significant, the shifting cost should be minimized due to the fact that it increases the labor costs and impair the client satisfaction.

At each port, a very large number of containers can be loaded, unloaded or repositioned. Despite the fact that such container movement plans reduce the transportation cost per container, it presents a difficult operational problem known as the Container Stowage Problem (CSP) (Avriel *et al*., 1998). Avriel *et al.* (2000) proved that the stowage planning problem is NP-complete by showing that the stowage problem is linked to the circle graphs coloring problem that is known to be NP-complete which implies that it is very unlikely to guarantee finding an optimal solution in a feasible processing time. Since, the 1970's, the container stowage planning problem has been studied by many researchers (Aslidis, 1989; Vis and Koster, 2003; Steenken *et al*., 2004; Günther and Kim, 2006; Stahlbock and Voβ, 2008). A lot of researches are mainly concentrated on the container loading problem that can be modeled as a combinatorial optimization problem (Aslidis 1989; Wilson and Roach, 1999; Wilson *et al*., 2001). A stowage plan involves the placement of a container at a ship slot which is described by a combination of the row number, bay number and tier number. Some of the main objectives of a good stowage plan are reducing handling time, guaranteeing stability, conforming ship stress limits and maximizing quay crane utilization (Wilson and Roach, 2000). Avriel *et al.* (1998) focus on stowage planning in consideration of minimizing the number of shifts. Haghani and Kaisar suggest a mixed integer program for modeling loading plans in order to minimize the time that a vessel spends at port and the container handling or manipulating cost which is strongly influenced by the number of shiftings caused by a bad stowage plan of containers. Dubrovsky *et al.* (2002) use a genetic algorithm for solving the stowage planning problem to minimize the number of container movements. Delgado *et al*. (2009) applied Constraints Programming (CP) to formulate the stowage planning problem. Moreover, a recent research about the stowage stack minimization problem with zero rehandle constraint was studied by Wang *et al.* (2014), it aims to find a minimum number of stacks needed to satisfy all the containers transportation in a multi-port voyage without facing container movements. Also, some papers focus on how to prevent shiftings by recommending methods to properly locate incoming containers in a container stack (Dekker *et al*., 2006; Casey and Kozan, 2012).

Simultaneously with these studies, the container stowage problem relates to several other problems in the literature. We refer to the berth allocation problem. (Cordeau *et al*., 2005). We mention as well the dynamic container relocation problem (Akyuz and Lee, 2014) and the ship routing and scheduling problem (Christiansen *et al*., 2004). Thus, for a more detailed discussion of each of these fields, we mention (Steenken *et al*., 2004; Stahlbock and Voβ, 2008; Douma *et al*., 2009).

Overview of the traveling salesman problem: The TSP is located at the heart of routing problems and listed as one of the most well studied, important and popular problems in combinatorial optimization. Moreover, according to its

simplicity and comprehensibility, TSP can model various other important problems and despite its clarity, it is extremely challenging and has motivated numerous publications dedicated to examine it more effectively. The classical TSP can be simply defined as finding the shortest tour between a set of cities which covers all cities each exactly once. The problem is NP-complete thus by increasing the number of cities, the computation time of optimal solution increases drastically (Garey and Johnson, 1979).

Since, its introduction, many different extended versions of the TSP have been studied. More and more constraints which make the resulting problems align better with real-life applications are being associated with the classical TSP. These extended versions include the pickup and delivery TSP with LIFO loading where visiting a delivery node is possible only if the load to be delivered is located at the top of the stack (Carrabs *et al*., 2007; Cordeau *et al*., 2010) with FIFO loading (Erdogan *et al*., 2009) and with multiple stacks (Petersen and Madsen, 2009) have been newly introduced as well as vehicle routing problems with loading issues (Iori and Martello, 2010).

In the classical version of the problem there is a fixed cost associated with each arc for traveling on this arc or for serving it. To our knowledge, relatively little effort has been devoted to the study of the TSP with a non-constant service cost. Most related works can be found in (Tagmouti *et al*., 2011) where the service cost on a required arc is a function of the time of beginning of service. Moreover, another research was presented by Tagmouti *et al*. (2007). In their research, they studied an arc routing problem with capacity constraints and time-dependent service costs where a subset of arcs must be serviced at a cost that depends on the time of beginning of service such that the cost is given as a piecewise linear function of time. Also, Wiel and Sahinidis (1996) presented an algorithm for solving the time-dependent traveling-salesman problem in which the cost of travel between two cities depends on the distance between the cities and the position of the transition in the tour. Whereas, Ichoua *et al*. (2003) proposed a time-dependent model for a vehicle routing problem with time windows, based on time-dependent travel speeds which satisfies the first in first out assumption.

There is a very large set of works in the literature on the TSP, we refer the interested reader to the surveys provided by Bektas (2006). Bigras *et al*. (2008), Iori and Martello (2010). To our knowledge, the ShTSP has not yet been studied in literature but several of other variants that include both pickups and deliveries have been investigated such as the traveling salesman problem with pickups, deliveries and handling costs (Battarra *et al*., 2010; Erdogan *et al*., 2012) which is a new variant of the one-to-many-to-one single vehicle pickup and delivery problems that incorporates the handling cost incurred when rearranging the load at the customer locations where each customer requires a pickup service, a delivery service or both, i.e., two types of items are considered those transported from the depot to customers and those transported from customers to the depot. This implies that the vehicle leaves the depot carrying all the deliveries, visits each customer once and returns to the depot carrying all pickups while respecting the capacity constraints. Other variants are the pickup and delivery problem with time windows and last in, first out loading (Cherkesly *et al*., 2014). The LIFO policy means that when a pickup point is visited, the collected item is positioned on top of a stack and can only be delivered if it is in that position. This rule ensures that no handling is required prior to unloading an item from a vehicle, the pickup and delivery problem with time windows and multiple stacks (Cherkesly *et al*., 2016) which prohibits additional handling operations as well, the pickup and delivery traveling salesman problem with handling costs (Veenstra *et al*., 2017a) where a single vehicle has to transport loads from many origins to destinations. Loading and unloading of the vehicle is operated in a LIFO policy and a penalty cost is associated with each additional handling operation and the pickup and delivery problem with time windows and handling operations (Veenstra *et al*., 2017b) where two different rehandling policies were defined. The first one only allows compulsory rehandling and the second one allows compulsory rehandling and preventive rehandling, i.e., all items can be rehandled at once.

MATERIALS AND METHODS

Problem statement and definition: This study addresses a new variant of the TSP that is combined with the ShP. Given one vehicle to deliver products stowed in one stack with a predefined order according to their fragility, weight and stability such that a number of products must be removed in order to reach products below them. Moreover, each product has a single client and each client has a single product.

From a graph theory point of view, the ShTSP can be formally defined on a complete graph as: let G = (V, A) be a complete graph where $V = \{c_0, c_1, ..., c_N\}$ is the vertex set such that all nodes are connected by arcs. Vertex c_0 represents the depot where the vehicle is initially located. The products are already stowed in it according to a predefined order. The remaining vertices $\{c_1, ..., c_N\}$

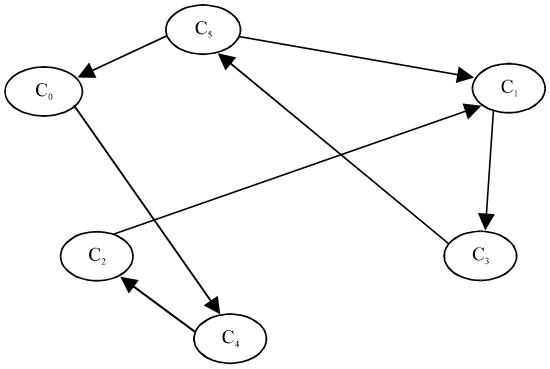

Fig. 1: A tour example of the vehicle

represent clients that must be served, each client c_i has a single product i. A non-negative value d_{ij} is assigned for each arc (i, j) to represents the distance cost and a non-negative value s_{ij} represents the shifting cost between each clients c_i and c_j. We note that $d_{ij} = d_{ji}$ and $s_{ij} = s_{ji}$.

For the encoding of the ShTSP's solutions, we have used the path representation shown in Fig. 1. In this way, each solution is encoded as a permutation of clients $c_i \forall i \in \{1, ..., N\}$; A sequence of the clients to be served starting and finishing at the depot c_0. Figure 1 shows a solution (c_0, c_4, c_2, c_1, c_3, c_5, c_0) with 5 clients.

Our aim is to seek an optimal tour that takes account of unloading and reloading of the products placed in the vehicle according to a predefined order. Our problem naturally arises when storage unit has only a single access located at the top and works like a stack; it is especially applicable when the shifting cost (i.e., unloading and reloading the blocked products) is comparable to that of the extra traveling distance caused by the LIFO policy of the stack.

To explain our problem, we consider a vehicle which has a vertical stack consisting of a finite set I = {1, ..., N} of N placement levels (Fig. 2), each level $i \in l$ represents a product i to be delivered identified by its unique destination c_i. The vehicle loading plan is known beforehand and is given according to the fragility, weight and stability of each product. The vehicle can only be accessed from above so the unloading of each product is performed according to the LIFO policy.

As mentioned before, we assume that the product $I \in l$ is intended for the single client c_i, therefore, at each subsequent clients requiring a delivery to serve the client c_i, we first have to unload all remaining products j (not yet served) such as $j<i$ ($i, j \in l$) which means discharge all products that are obstructing the unloading of the concerned product I. Then, after delivering the concerned product, the unloaded obstructing products must be

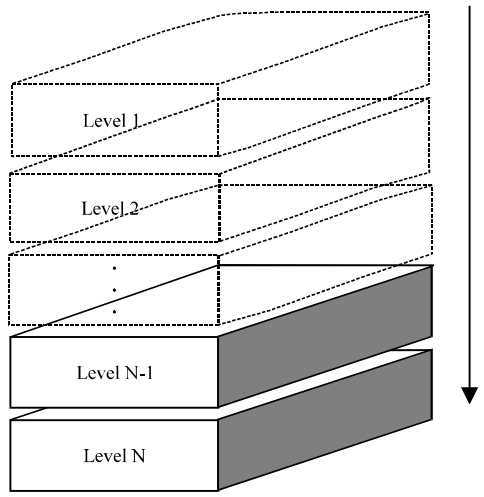

Fig. 2: Illustration of the stack

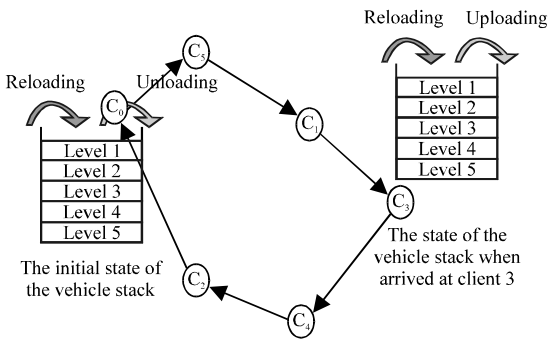

Fig. 3: Example of a feasible tour of the ShTSP

reloaded again into the vehicle to continue the tour (Fig. 3). Each of these additional movements is called a shifting. More specifically a shifting is defined as the temporary removal of products which are arranged above the concerned product and their placement once again in the vehicle.

For illustration, we consider a small example where the vehicle needs to serve five clients. Figure 3 shows a tour solution which starts with the client c_5, must therefore, discharge products at level 1-4 then serve the client c_5 and subsequently reload the products 1-4 in the same order as before to continue the tour and so on until the end of the tour (Fig. 4).

To simplify, we define all costs in terms of time unit (minutes). We assume that each arc is valued by the travel time between the two corresponding clients as shown in Fig. 5 and each of the unloading (u) and reloading (r) costs of product i, $\forall i \in \{1, ..., N\}$ is equal to 3 min. The total shifting cost of the tour will be equal to 51 min as calculated in Fig. 4 where each case represents the state of the vehicle when arrived at each client following the

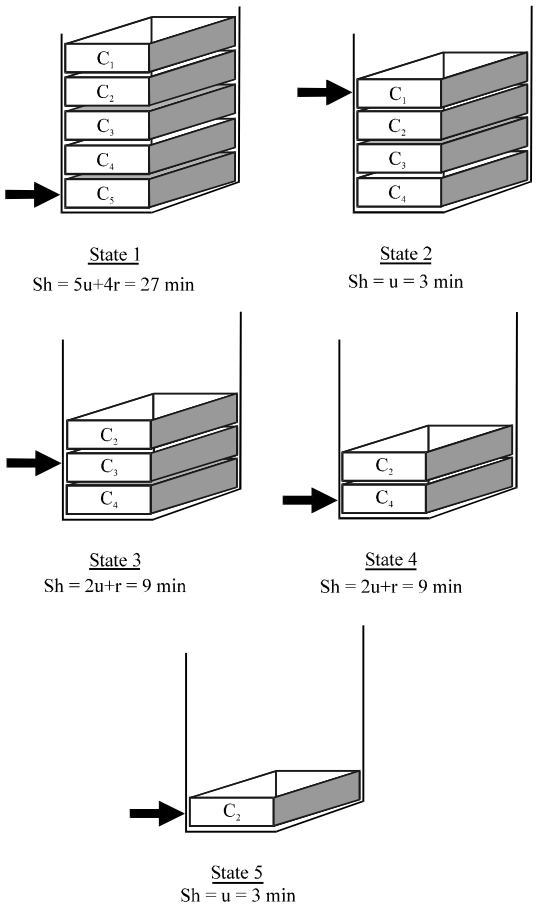

Fig. 4: Illustration of each state of the stack throughout the tour

same order of the tour presented in Fig. 3. In this example, we can see that there are several shiftings to be made when unloading the desired product at each client.

Note that minimizing both the total distance traveled and the number of shiftings made do not vary generally in the same way. Indeed, minimizing the distance cost of the tour could increase its total shifting cost and vice versa. In Fig. 5, we have two possible solutions a and b that vary in a different way. Compared with the tour b, the tour a represents a very good solution in terms of travel time (the minimal distance) where $Tarvel_{time\ (a)} = 3+7+5+6+2+4 = 27$ min and $Tarvel_{time\ (b)} = 13+17+7+6+10+3 = 56$ min. On the other hand, it represents a less good solution in terms of shifting costs compared with the tour b which describes, this time, the best solution in terms of shifting time where $Shifting_{time(b)} = 5\times3 = 15$ min and $Shifting_{time(a)} = 51$ min (Fig. 4).

In most times, service costs are either ignored or assumed to be constant. However, in practice, it can easily be observed that service costs vary according to

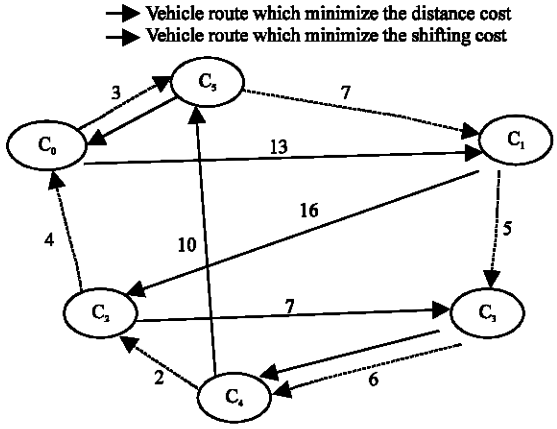

Fig. 5: The two objectives (minimizing the distance and shiftings costs) do not vary in the same way

several factors which naturally depend on the order of the customers visited in each node. In our problem, the service cost required at each client is not fixed a priori but depends on the position of this client in the tour. Note that sometimes, the shifting cost is larger than the distance cost because it takes more time and effort to service an arc than to simply travel along the arc. Thus, the primary contribution of this resarch is to introduce a new variant of the traditional TSP applied in the ShP that addresses the challenge of determining the vehicle's optimal tour takes account of these additional movements of the products by unloading and reloading them at each client. Therefore, our goal is to minimize the total tour costs of a single vehicle with a single stack including the total travel distance and the total service cost which is represented by the total number of shiftings in this tour caused by the unloading and reloading of all the blocked products in the stack. To the best of the resarchers knowledge, the ShTSP represents a new variant of the TSP and it has not been handled previously in the literature as the same way as introduced in this research.

Problem formulation: In this study, we present a mixed nonlinear programming formulation to solve the ShTSP. Before that we introduce some assumptions and terminology which will be used in formulation later.

The basic assumptions used in this resarch are listed below: the vehicle loading plan is known before hand and is given according to the stability and weight of each product. The vehicle has exactly one vertical stack consisting of a number of placement levels. The unloading order of the products in the vehicle must be done either from the top or from the side. In this study, the vehicle can only be accessed through the top, so that, it will follow a last-in-first-out unloading policy. Each

client asks for the delivery of one product. In other words, the number of clients to be served is the same as that of products stowed in the vehicle. Only one product may be stowed in each level of the stack and must be retrieved to its corresponding client. The following notations are used throughout the study to formulate the model:

- N: the number of clients to be visited (equals to the number of products to deliver)
- M: a positive constant sufficiently large
- α: a small positive constant
- d_{ij} the distance between clients c_i and c_j
- u_i the unloading cost of product i
- r_i the reloading cost of product i
- $b_{ij} = \begin{cases} 1 & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}$

With these notations in hand, a mixed nonlinear programming formulation of the ShTSP may be constructed with the introduction of the variables, for i, $j \in \{0, ..., N\}$ we define:

$$x_{ij} = \begin{cases} 1 & \text{if the product is delivered immediately} \\ & \text{after the product i} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if the product is delivered} \\ & \text{before the product j} \\ 0 & \text{otherwise} \end{cases}$$

Note that y_{ij} if the product i is delivered not necessary "Immediately" before the product j:

S_{ij} = The shifting cost of product j knowing that the vehicle has just served client c_i

Where:

$$S_{oj} = \sum_{i=1}^{j} u_i + \sum_{i=1}^{j-1} r_i \qquad \forall j \in \{1,...,N\} \tag{1}$$

$$S_{io} = 0 \qquad \forall i \in \{1,...,N\} \tag{2}$$

Equation 1 defines the shifting cost at the beginning of the tour which is defined as the unloading and reloading costs of all products stowed on top of product j. However, the second formula indicates that at the end of the tour there is no shifting cost i.e. there is no unloading or reloading operation. And for $i \neq 0$ and $j \neq 0$ the shifting cost s_{ij} can be found as follow:

$$\begin{aligned} S_{ij} = (1-b_{ij}) \Bigg[& \sum_{\substack{k=0 \\ k\neq i \\ k\neq j}}^{N} S_{ki} x_{ki} - u_i + u_j \\ & + \sum_{k=i+1}^{j-1} (u_k + r_k)(1-y_{kj}) \Bigg] \\ & + b_{ij} \Bigg[\left(s_{0i} - \sum_{k=j+1}^{i-1} (u_k + r_k) \right) x_{0i} \\ & + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} \Big(s_{ki} \\ & - \sum_{v=j+1}^{i-1} (u_v + r_v)(1-y_{vj}) \Big) x_{ki} - u_i - r_j \Bigg] \end{aligned} \tag{3}$$

$$\forall i, j \in \{1,...,N\} \text{ with } i \neq j+1, j \neq i+1 \text{ and } i \neq j$$

$$S_{ij} = \sum_{\substack{k=0 \\ k=i,j}}^{N} s_{ki} x_{ki} - u_i - r_i \qquad \forall i, j \in \{1,...,N\} \text{ such as } i = j+1 \tag{4}$$

$$S_{ij} = \sum_{\substack{k=0 \\ k=i,j}}^{N} s_{ki} x_{ki} - u_i - u_j \qquad \forall i, j \in \{1,...,N\} \text{ such as } j = i+1 \tag{5}$$

Equation 3-5 are given in a recursively form they indicate the shifting cost of product j knowing that the vehicle has just visited client c_i which means, the traveler has just delivered product i in a way to take account of the history of all clients visited before c_i according to the order found until the current moment. In other words, they give the force of eliminating all the products that are already delivered before the product j. Where of, we proposed a new way of modeling shifting costs required to deliver every product i to its corresponding client c_i which are not fixed a priori yet depend on the order of clients that were visited before c_i. The key idea of this recursively form is the specification of knowing in every moment the history of every non-complete tour:

$$\alpha x_{ij} \leq s_{ij} \leq M x_{ij} \quad \forall i \in \{0,...,N\}, \; \forall j \in \{1,...,N\}, \; i \neq j \tag{6a}$$

To explain constraints Eq. 6a, we suppose that the vehicle visits client c_j immediately after client c_i in this case $x_{ij} = 1$, so, the shifting cost of unloading the product j is greater than a positive constant α, otherwise the shifting cost is equal to zero. Furthermore, considering the stack levels presentation showed in Fig. 2 these positive

constants α and M can be defined, more precisely as the best case scenario and the worst case scenario of the shifting solutions, respectively such as $\alpha = \min_{i, j \in \{0, ..., N\}} \{S_{ij}\}$ which means the minimal cost of unloading the upper-most product of the stack in other words $\alpha = \min_{k \in \{0, ..., N\}} \{u_k\}$ whereas $M = \alpha = \max_{i, j \in \{0, ..., N\}} \{S_{ij}\}$ which means the cost of unloading the lower-most product of the stack such that all the products above it are still not served yet in simple terms $M = s_{ON}$. Hence, Eq. 6a can be replaced by the following Eq. 6b:

$$x_{ij} \min_{k \in \{1,...,N\}} \{u_k\} \leq s_{ij} \leq x_{ij} s_{oN} \quad \forall i \in \{0,...,N\}, \quad \forall j \in \{1,...,N\}, \quad i \neq j \tag{6b}$$

$$x_{ij} + x_{ij} \leq 1 \qquad \forall i, j \in \{0,...,N\}, \quad i < j \tag{7}$$

Constraints Eq. 7 indicate that if product j is delivered immediately after product i then we must have x_{ji} which eliminates the opposite case:

$$y_{ij} + y_{ji} = 1 \qquad \forall i, j \in \{0,...,N\}, \quad i<j \tag{8}$$

Constraints Eq. 8 capture the fact that for each pair of clients c_i and c_j, the vehicle will serve one of these clients before the other. In other words, if client c_j precedes client c_j on a tour then c_j cannot precedes c_i on that same tour:

$$y_{ij} \geq x_{ij} = 1 \qquad \forall i, j \in \{0,...,N\}, \quad i \neq j \tag{9}$$

Constraints Eq. 9 define the connection between the decision variables by ensuring the fact that if client c_i precedes directly client c_j on a tour then c_j cannot precede c_i on that sametour:

$$y_{ij} \geq y_{ik} + y_{kj} - 1 \qquad \forall i, j, k \in \{0,...,N\}, \quad i \neq j, \; j \neq k, k \neq i \tag{10}$$

Constraints Eq. 10 enforce that client c_i must precede client c_j if client c_k precedes client c_j and client c_i precedes client c_k on the tour of the vehicle. They ensure the order of unloading the products of clients c_i, c_j and c_k:

$$\sum_{\substack{i=0 \\ i \neq j}}^{N} x_{ij} = 1 \qquad \forall j \in \{0,...,N\} \tag{11}$$

$$\sum_{\substack{i=0 \\ j \neq i}}^{N} x_{ij} = 1 \qquad \forall j \in \{0,...,N\} \tag{12}$$

Constraints Eq. 11 and 12 are the degree constraints they ensure that each client is visited exactly once by the vehicle:

$$\sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1 \qquad \forall Q \text{subset of } \{1, ..., N\} \tag{13}$$

Constraints Eq. 13 are for breaking subtour problems:

$$x_{ij}, y_{ij} \in \{0,1\} \qquad \forall i, j \in \{0,...,N\} \tag{14}$$

$$S_{ij} \geq 0 \qquad \forall i, j \in \{0,...,N\} \tag{15}$$

And finally, constraints Eq. 14 and 15 specify the variable definitions. Note that constraints Eq. 15 can also indicate that partial service at every client of the tour is not allowed. Thus, the objective function of the mathematical model may be stated as:

$$\text{Min}\left(\sum_{\substack{i, j=0 \\ i \neq j}}^{N} x_{ij} d_{ij}, \sum_{\substack{i, j=0 \\ i \neq j}}^{N} s_{ij} \right) \tag{16}$$

This objective function Eq. 16 aims to minimize the total tour cost of the vehicle which consists of minimizing two functions, the total distance traveled and the total shifting cost that is defined by the unloading and reloading costs of each product in every client of the tour.

RESULTS AND DISCUSSION

In this study, we adapt the Ant Colony metaheuristic (ACO) in two different ways to solve the ShTSP. We are exploiting a colony of ants in order to vary the order in which products are delivered to their corresponding destinations such that both the distance traveled and the placement's level of each product influence the quality of each solution obtained. First, we present a general adaptation of the ACO Algorithm adjusted to the ShTSP and then we solve our problem with a direct method which explores all the solutions exhaustively, on randomly generated test problems, to prove the efficiency of our ACO adaptation. Also, we focused on how to speed up the solution obtained by the sequential algorithm by proposing a parallel-ant colony algorithm adapted to our problem. Finally, numerical results applied to new instances generated from TSPLIB are shown for each algorithm.

Ant colony metaheuristic's principle: The ant colony algorithm is a stochastic process building a solution by

adding components to partial solutions. This process takes into account two factors, the first is a heuristic on the instance of the problem and the second is pheromone tracks that change dynamically to repeat the experience acquired by the agents. Ants thus, build solutions while moving on a graph G = (V , E) where V is the set of nodes and E connects the components of V. The problem constraints are implemented directly in the ants movement rules. Thereby, the general concept of this metaheuristic is based on the simulation of the behavior of several agents working together in search of a better solution using a natural way of communication that is the deposit pheromone.

Adaptation of ant colony metaheuristic: To solve the ShTSP, we adapt the metaheuristic of ant colony according to the following steps. First, each ant k will start from the depot c_0 and goes randomly to a client c_i, then we calculate the heuristic information $(\eta_{ij})_{j\in [0, ..., N]}$ according to what was chosen before as a client (i.e., the client c_i) using the following formula:

$$\eta(ij) = \begin{cases} \frac{1}{s_{ij}+d_{ij}} & \forall j \in L_i^k \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

Where L_i^k indicates the list of candidates (clients that are not yet visited) of the ant k at node i and s_{ij} is calculated according to the Eq. 4-6 shown in the formulation problem section.

This heuristic information is used to guide the choice of ants to near clients (in term of distance) of which indexes are as small as possible, i.e., the cost of unloading their corresponding product is small. We prefer client that has both the smallest traveling distance and the smallest shifting cost which means its corresponding product is placed in the highest levels. After that we calculate the probability of choosing the next client from the set of candidates of the chosen client c_i (in the previous step), according to the following Eq. 18:

$$p_{ij}^k = \begin{cases} \frac{(\tau_{ij})^{\alpha}.(\eta_{ij})^{\beta}}{\sum_{l\in L_i^k}(\tau_{il})^{\alpha}.(\eta_{il})^{\beta}} & \text{if } j \in L_i^k \\ 0 & \text{otherwise} \end{cases} \quad (18)$$

The 2 parameters α and β define the relative importance between the value of the pheromone τ_{ij} and the heuristic information η_{ij}.

The deposit of pheromones can greatly change the convergence mode of the algorithm. From a purely Naive point of view, we can completely deposit the same amount of pheromone on each path. The ants engaged in long paths will file fewer pheromones since they are able to try fewer paths. On the contrary, ants engaged on the shortest paths will rapidly try other paths. Naturally, the shortest paths will find more pheromones than others. However, we can use two methods for updating the pheromone value: we choose the best solution of all ants, and then we do an update of the $(\tau_{ij})_{i, j}$ associated to the best solution. After that we perform a global update of the pheromone for all arcs of the graph following the Eq. 19:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \quad (19)$$

Where:

ρ = A coefficient of evaporation chosen between [0, 1]

m = The number of ants in the colony

$\Delta\tau_{ij}^k$ is calculated according to the next equation:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{q}{L^k} & \text{if } (i,j) \text{ is part of the solution of ant k} \\ 0 & \text{otherwise} \end{cases}$$

Where:

L^k = The total cost of the solution

k and q = A fixed parameter of the algorithm

Algorithm 1 gives the skeleton of the adaptation of ant colony metaheuristic to solve the ShTSP.

Algorithm 1: ACO algorithm for the ShTSP:

```
Input
    m←number of ants
    iterNumber←number of iterations
    τ0←the initial amount of pheromone which is a positive
       real number
    iter←1
for each arc (i, j) do
  τij = τ0
end for
while (iter≤iterNumber) do
    for k = 1 until m do
        while (ant k has not completed the solution yet) do
            Construct the solution Sol^k(iter) according to
            formula 17 and 18
            Calculate the cost of Sol^k(iter)
        end while
    end for
    if a better solution was found then
    Update the best solution found
    end if
    for all arc (i, j) do
        Update the amount of pheromone τij according to formula 19
    end for
    iter←iter+1
end while
```

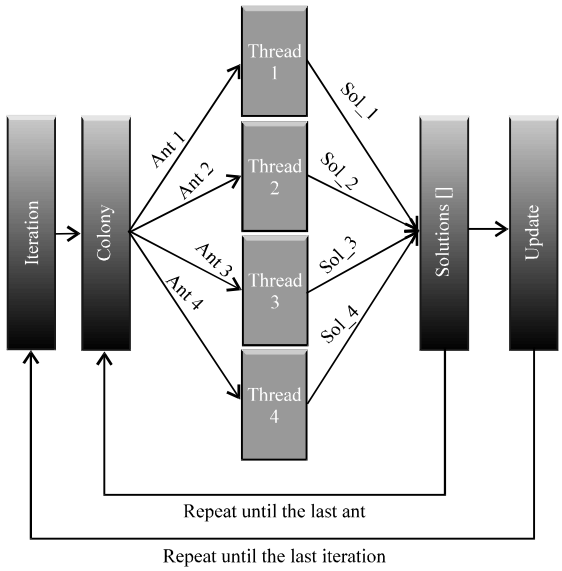

Fig. 6: The parallel-ACO algorithm

Direct algorithm: To evaluate our adaptation of ACO algorithm, we performed some tests and compared them by solving random instances with a direct algorithm which is an exact method based on combinatoricslib that consists on a very simple java library to generate permutations, combinations and other combinatorial sequences (Algorithm 2).

Algorithm 2; Direct algorithm for the ShTSP:

```
Search for all possible permutations of all clients, starting by the depot 0,
using the combinatoricslib
for each permutation p do
    Calculate the cost of p
    if a better solution was found then
        Update the best solution found: p*
    end if
end for
```

The main disadvantage of this exact method is that it reaches its limit with only a slightly larger number of clients, even if a more powerful machine was used.

Adaptation of parallel-ant colony algorithm: While it is important to achieve good solutions, computational time is a valuable indicator that must be considered in real-world applications. In this study, we propose a new adaptation of the ACO algorithm by using the concept of parallelism as shown in Fig. 6.

The parallelism in our context aims to divide the computation time of each colony by launching a process for each ant, it serves to optimize the use of computational resources in order to optimize the efficiency of the algorithm.

Algorithm 3; Parallel-ACO algorithm for the ShTSP:

```
Input
    m←number of ants
    iterNumber←number of iterations
        τ0←the initial amount of pheromone
    iter←1
for all arc (i,j) do
        τij = τ0
end for
for iter = 1 until iter Number do
    Parallel Construct solutions Sol (iter) [p] for a set of p
            ant
    Until the last ant
    if a better solution was found then
    Update the best solution found: Sol*
    end if
    for all arc (i, j) do
            Update the amount of pheromone
    end for
end for
```

Computational results: In this study, we report the numerical results of our problem solved with a direct method, ACO algorithm and parallel-ACO algorithm. The resolution approaches have been implemented with Java using NetBeans 8.0. All experiments were conducted on Intel® Core™ i5-4570 CPU @ 3.20 GHz.

Because there is no suitable set of benchmark problems in the literature which combines the ShP with the TSP, we consider generating some instances with different properties. The ShTSP makes sense in contexts where distance costs and total shiftings costs are in the same average. We have therefore, tested our algorithms on instances exhibiting this characteristic. We have generated new benchmarks, based on the TSPLIB in which the shifting costs are comparable to the routing costs. Each set of problems is generated with different unloading/reloading costs which are set at 3, 7, 10, 12, 15, 20, 25 and 30 time units per product. Note that one unit of time is equivalent to one unit of Euclidean distance.

TSPLIB is available athttp://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. The specific information: the problem names (Inst_TSPLIB) along with the number of clients (|V|), the problem type (Type), the unloading and reloading costs (U×R) and the new problem names (Inst_Sh) about each instance used in this study is given in two tables (Table 1 and 2).

Table 1 explains the small/medium sized instances. The first column indicates the name of instances in the TSPLIB that were used to create new instances, Inst_Sh, adapted to our problem by adding the Unloading (U) and Reloading (R) costs which are stated in minutes and shown in the fourth column.

Table 2 explains the big sized instances, adapted to the present problem, following the same structure as Table 1. New additional instances were randomly generated to investigate the performance of our

adaptation of ACO algorithm and to compare its efficiency by using the direct method. These instances are divided into three classes, generated randomly and distinguished by the unloading and reloading cost of a single product in the vehicle:

Table 1: Explain instance information with small |V|

Inst_TSPLIB	|V|	Type	U×R	Inst_Sh
gr17	17	Matrix	15×15	gr17_sh15
			25×25	gr17_sh25
			30×30	gr17_sh30
gr21	21	Matrix	15×15	gr21_sh15
			25×25	gr21_sh25
			30×30	gr21_sh30
gr24	24	Matrix	7×7	gr24_sh7
			10×10	gr24_sh10
			12×12	gr24_sh12
fri26	26	Matrix	7×7	fri26_sh7
			10×10	fri26_sh10
swiss42	42	Matrix	3×3	swiss42_sh3
			7×7	swiss42_sh7
dantzig42	42	Matrix	3×3	dantzig42_sh3
			7×7	dantzig42_sh7
hk48	48	Matrix	15×15	hk48_sh15
			25×25	hk48_sh25
			30×30	hk48_sh30
gr48	48	Matrix	10×10	gr48_sh10
			12×12	gr48_sh12
			15×15	gr48_sh15
			25×25	gr48_sh25
eil51	51	EUD_2D	3×3	eil51_sh3
			7×7	eil51_sh7
berlin52	52	EUD_2D	15×15	berlin52_sh15
			25×25	berlin52_sh25
			30×30	berlin52_sh30

Table 2: Explain instance information with big |V|

Inst_TSPLIB	|V|	Type	UxR	Inst_Sh
st70	70	EUD_2D	3×3	st70_sh3
eil76	76	EUD_2D	3×3	eil76_sh3
rat99	99	EUD_2D	3×3	rat99_sh3
kroA100	100	EUD_2D	3×3	kroA100_sh3
			7×7	kroA100_sh7
			10×10	kroA100_sh10
			12×12	kroA100_sh12
			15×15	kroA100_sh15
			20×20	kroA100_sh20
kroB100	100	EUD_2D	3×3	kroB100_sh3
			7×7	kroB100_sh7
			10×10	kroB100_sh10
			12×12	kroB100_sh12
			15×15	kroB100_sh15

- c_1: the unloading and reloading costs of a single product U = R = 5 min
- c_2: the unloading and reloading costs of a single product U = R = 10 min
- c_3: the unloading and reloading costs of a single product U = R = 15 min

Each class consists of three groups that are distinguished by the number of clients to be served (|V|= 5, |V| = 7 and |V| = 10).

The results of the c_1-c_3 instances are presented together with a description of the 3 classes in Table 3. The first three columns specify the attributes of groups of instances: the class (class), the number of clients per instance (|V|) and the uniform distribution interval [min, max] which gives the distance values of the graph of each group. Each row represents the results obtained for instances that constitute a group in a class. The direct method gives the solutions presented in the direct method's column. We give the explicit tour path in the fourth column (DTourPath) (note that in the present research, we were limited on a path which will be presented as a tour without considering the return to the depot), the shifting cost in the fifth column (DSh_C, the traveling cost in the sixth column (DTrav_C) and the total tour cost of the solution in the seventh column (DTotal_C). While the results of the adaptation of the ACO algorithm on the same generated instances are summarized in the our method (ACO)'s column, we give the explicit tour path (AtourPath), the total tour cost of the solution (ATotal_C) and the CPU time corresponding to the execution of the algorithm in milliseconds (ACPU). Finally, we show the deviation of our adaptation ACO from the exact method in the last column (Gap).

Referring to Table 3, we can assume that the ACO algorithm gives very good results for instances of small sizes. Therefore, we can be sure that our algorithm will give very satisfying solutions for instances of larger sizes.

While it is important to achieve good solutions, computational time is a valuable indicator that must be

Table 3: Numerical results of c_1, c_2 and c_3 instances found by ACO's adaptation and compared with the direct method

			Direct method				Our method (ACO)			
Class	|V|	[min, max]	Dtourpath	Dsh_C	Dtrav_C	Dtotal_C	Atourpath	ATotal_C	ACPU (msec)	Gap
c1	5	[3,50]	[0, 1, 2, 3, 4]	20	89	109	[0, 1, 2, 3, 4]	109	46	**0.00**
	7	[5,70]	[0, 1, 6, 2, 4, 3, 5]	80	97	177	[0, 1, 2, 4, 3, 5, 6]	186	54	9.00
	10	[7,100]	[0, 2, 6, 1, 9, 3, 8, 4, 5, 7]	175	180	355	[0, 2, 4, 1, 6, 3, 8, 9, 5, 7]	387	94	32.00
c2	5	[9,90]	[0, 1, 3, 2, 4]	60	131	191	[0, 1, 3, 2, 4]	191	39	**0.00**
	7	[10,140]	[0, 4, 6, 1, 3, 2, 5]	220	222	442	[0, 4, 6, 1, 3, 2, 5]	442	54	**0.00**
	10	[15,200]	[0, 1, 6, 2, 3, 5, 4, 7, 8, 9]	190	457	647	[0, 1, 6, 2, 3, 5, 4, 7, 8, 9]	647	102	**0.00**
c3	5	[10,140]	[0, 1, 2, 4, 3]	90	234	324	[0, 1, 2, 4, 3]	324	31	**0.00**
	7	[20,200]	[0, 2, 1, 4, 3, 5, 6]	150	312	462	[0, 2, 1, 4, 3, 5, 6]	462	54	**0.00**
	10	[30,280]	[0, 1, 2, 3, 7, 4, 5, 8, 6, 9]	255	700	955	[0, 1, 2, 3, 7, 4, 5, 8, 6, 9]	955	101	**0.00**

Bold values are significant values

Table 4: Numerical results of the adaptation of ACO and parallel ACO on instances of Table 1

	ACO				Parallel-ACO			
	50 ants		100 ants		50 ants		100 ants	
Instance	TCost	CPU (msec)	TCost	CPU (msec)	TCost	CPU (msec)	TCost	CPU (msec)
gr17_sh15	3374	203	3374	391	3374	**15**	3374	**30**
gr17_sh25	3878	203	3878	281	3878	**15**	3878	**28**
gr17_sh30	4231	141	4231	297	4231	**15**	4231	**28**
gr21_sh15	4927	219	4927	422	4927	**22**	4927	**44**
gr21_sh25	5531	203	5531	422	5531	**22**	5531	**44**
gr21_sh30	5858	218	5858	422	5858	**21**	5858	**43**
gr24_sh7	2431	266	2431	532	2431	**30**	2431	**59**
gr24_sh10	2847	281	2847	531	2847	**30**	2847	**59**
gr24_sh12	3084	266	3084	515	3084	**29**	3084	**59**
fri26_sh7	1134	297	1134	593	1134	**33**	1134	**63**
fri26_sh10	1209	297	1209	594	1209	**33**	1209	**63**
swiss42_sh3	2630	828	2630	1641	2630	**96**	2630	**190**
swiss42_sh7	3019	843	3019	1625	3019	**95**	3019	**187**
dantzig42_sh3	839	812	839	1641	839	**94**	839	**187**
dantzig42_sh7	966	813	966	1640	966	**94**	966	**187**
hk48_sh15	**23737**	**1047**	**23060**	**2109**	**23144**	**131**	**23060**	**263**
hk48_sh25	26063	1062	26063	2079	26063	**132**	26063	**262**
hk48_sh30	27942	1047	27942	2078	27942	**132**	27942	**261**
gr48_sh10	11488	1094	11488	2157	11488	**127**	11488	**253**
gr48_sh12	**11680**	**1109**	**12050**	**2187**	**11680**	**127**	**11680**	**253**
gr48_sh15	12245	1078	12245	2157	12245	**126**	12245	**252**
gr48_sh25	**15173**	**1078**	**15173**	**2187**	**15334**	**126**	**15173**	**251**
eil51_sh3	1293	1203	1293	2344	1293	**142**	1293	**282**
eil51_sh7	1673	1188	1673	2359	1673	**142**	1673	**281**
berlin52_sh15	19419	1218	19419	2531	19419	**156**	19419	**308**
berlin52_sh25	19600	1266	19600	2516	19600	**156**	19600	**307**
berlin52_sh30	19792	1250	19792	2516	19792	**155**	19792	**309**

Bold values are large instances

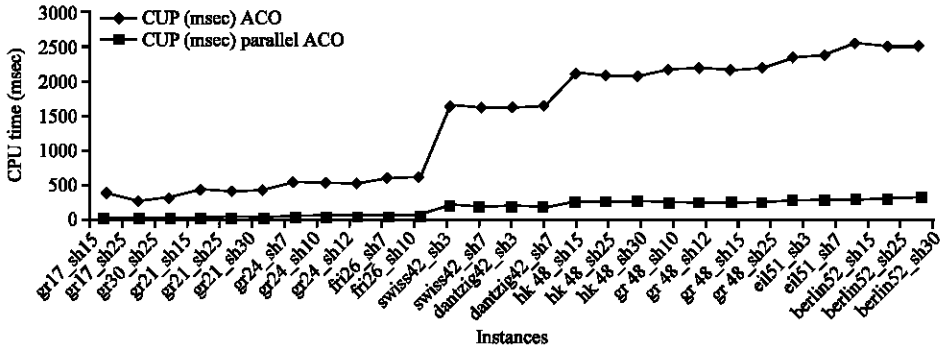

Fig. 7: Comparison of the CPU time of ACO and parallel ACO both adapted to instances of Table 1 with 100 ants using four threads

considered in real-world applications. In the following Table 4, we show the numerical results of the adaptation of both the ACO algorithm and the parallel-ACO algorithm obtained by resolving our new instances shown in Table 1. Whereas in Table 5, we give numerical results for l arger instances which are shown in Table 2. Note that each row represents the best solution in 50 runs. In the ACO column as well as in the parallel-ACO column, we give the results obtained by using both, 50 and 100 ants, we show the total cost of the solution (TCost) and the CPU time by millisecond (CPU).

Referring to the results, we can confirm that for small instances when the number of ants increases in both the ACO and parallel-ACO we obtain better results in terms of TCost but only for a few instances (which are represented in bold in Table 4: hk48_shl5, gr48_shl2 and gr48_sh25). However, in Table 5 which represents larger instances, we obtain better results for almost all instances (represented in bold in Table 5). Moreover, when the number of ants increases, the performance of parallel-ACO increases as well in terms of execution CPU time for both smaller and larger instances.

In addition, we give a comparison in terms of CPU time of ACO and parallel-ACO, both adapted to instances of Table 1 by using 100 ants (Fig. 7). Note that these results were conducted on Intel® Core™ i5-4570 CPU @

Table 5: Numerical results of the adaptation of ACO and parallel ACO on instances of Table 2

	ACO				-bfgParallel-ACO			
	50 ants		100 ants		50 ants		100 ants	
Instances	TCost	CPU (msec)	TCost	CPU (msec)	TCost	CPU (msec)	TCost	CPU (msec)
st70_sh3	2568	3406	2568	6905	2568	**661**	2595	**1320**
eil76_sh3	2133	4125	2133	8187	2133	**785**	2133	**1573**
rat99_sh3	2792	7016	2792	14033	2792	**1390**	2792	**2777**
kroA100_sh3	38888	7187	38888	14015	38888	**1439**	38888	**2882**
kroA100_sh7	**53538**	**7188**	**53538**	**13805**	**53616**	**1434**	**53538**	**2871**
kroA100_sh10	**57851**	**7234**	**57851**	**13985**	**58839**	**1432**	**57851**	**2863**
kroA100_sh12	**62236**	**7188**	**61855**	**14187**	**61855**	**1431**	**61198**	**2862**
kroA100_sh15	**66450**	**7188**	**66773**	**14126**	**66450**	**1430**	**66280**	**2863**
kroA100_sh20	71414	7312	71414	14328	71414	**1430**	72232	**2856**
kroB100_sh3	40362	7251	40362	14469	40362	**1437**	40362	**2870**
kroB100_sh7	**49989**	**7203**	**49836**	**14375**	**49836**	**1434**	**49836**	**2865**
kroB100_sh10	**59727**	**7250**	**59396**	**14391**	**59396**	**1433**	**59396**	**2860**
kroB100_sh12	**61399**	**7219**	**61399**	**14391**	**61624**	**1431**	**61399**	**2859**
kroB100_sh15	66425	7203	66425	14422	66425	**1431**	66425	**2856**
kroB100_sh20	**73814**	**7203**	**73323**	**14312**	**73814**	**1429**	**73323**	**2853**
kroC100_sh3	37593	7234	37593	14485	37593	**1436**	37593	**2869**
kroC100_sh7	**49428**	**7219**	**48990**	**14406**	**49428**	**1433**	**48990**	**2864**
kroC100_sh10	56505	7219	56505	14484	56505	**1432**	56505	**2860**
kroC100_sh12	**58923**	**7203**	**57266**	**14407**	**58923**	**1431**	**57266**	**2859**
kroC100_sh15	**63884**	**7203**	**63396**	**14406**	**63884**	**1430**	**63396**	**2856**
kroC100_sh20	70809	7235	70809	14360	70809	**1430**	70809	**2855**
kroD100_sh3	**39257**	**7219**	**38858**	**14375**	**39257**	**1435**	**38858**	**2870**
kroD100_sh7	**52028**	**7219**	**51424**	**14344**	**52028**	**1433**	**51424**	**2862**
kroD100_sh10	58029	7234	58029	14406	58029	**1432**	58029	**2859**
kroD100_sh12	61187	7219	61187	14391	61187	**1431**	61187	**2859**
kroD100_sh15	**64584**	**7203**	**64033**	**14344**	**64584**	**1430**	**64033**	**2856**
kroD100_sh20	71203	7219	71203	14328	71203	**1428**	71203	**2853**
kroE100_sh3	**40904**	**7204**	**40852**	**14375**	**40904**	**1438**	**40852**	**2871**
kroE100_sh7	**50961**	**7234**	**50255**	**14375**	**50961**	**1433**	**50255**	**2863**
kroE100_sh10	**59157**	**7219**	**59157**	**14407**	**59157**	**1430**	**59067**	**2860**
kroE100_sh12	**64329**	**7265**	**63541**	**14344**	**64329**	**1430**	**63541**	**2860**
kroE100_sh15	**65348**	**7235**	**63621**	**14344**	**65348**	**1433**	**63621**	**2859**
kroE100_sh20	**74006**	**7203**	**73853**	**14390**	**73853**	**1430**	**73853**	**2855**
eil101_sh3	**2315**	**7343**	**2236**	**14625**	**2315**	**1453**	**2236**	**2902**
lin105_sh3	**23193**	**7984**	**23151**	**15922**	**23193**	**1587**	**23151**	**3173**
lin105_sh7	**29317**	**7954**	**28916**	**15875**	**29317**	**1587**	**28916**	**3171**
lin105_sh10	29737	7953	29737	15875	29737	**1585**	29737	**3167**
lin105_sh12	**31151**	**7953**	**30268**	**15891**	**31151**	**1584**	**30268**	**3165**
lin105_sh15	**34907**	**7938**	**34482**	**15906**	**34907**	**1584**	**34482**	**3165**

Bold values are adaptation of results on instances of Table 2

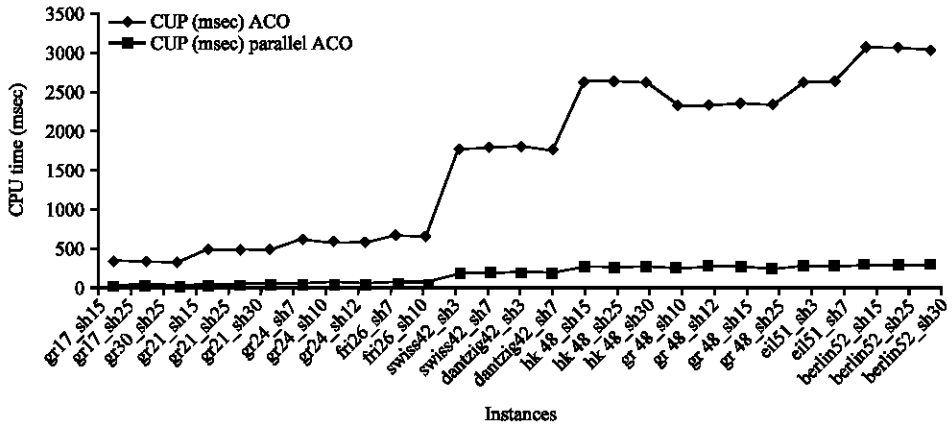

Fig. 8: Comparison of the CPU time of ACO and Parallel ACO both adapted to instances of Table 1 with 100 ants using eight threads

3.20 GHz which has four threads. Whereas in Fig. 8, we tested our algorithms in Intel® Core™ i7-4770 CPU @ 3.40 GHz which has eight threads. Both charts confirm the performance of the parallelism concept in our algorithm by showing the big gap between the CPU time of the simple version (ACO) and that of the parallel one (parallel-ACO).

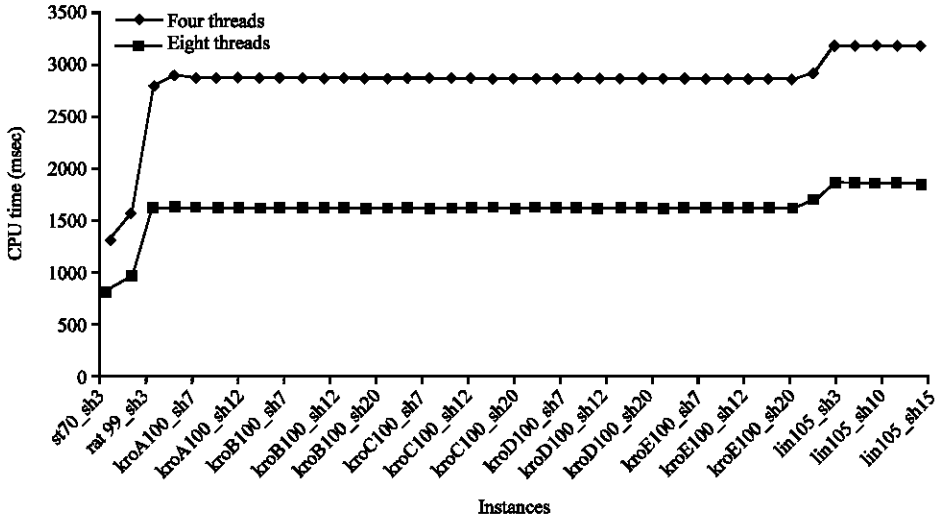

Fig. 9: Comparison of the CPU time (msec) of parallel ACO adapted to instances of Table 2 with 100 ants by using four threads and eight threads

In particular, a sharp decrease in the parallel ACO's computation time is observed starting from the instance swiss 42_sh3 with 42 clients, especially, when using eight threads.

To show the relation between the performance of the parallel-ACO algorithm and the number of threads, we plotted in Fig. 9, the progress of the CPU time of parallel-ACO adapted to instances of Table 2 with 100 ants by using both four threads and eight threads. We can confirm that the parallel-ACO algorithm shows more efficiency when executed in a machine with more threads.

To conclude and referring to all results given above, we can confirm that the parallel-ACO algorithm gives excellent results on average of the CPU time solutions for all instances compared with the ACO algorithm. It allows us to reduce the computational time by more than 80%, especially, for large sized instances. Furthermore, the variation of the CPU time is strongly correlated with the variation of the number of threads in the machine.

CONCLUSION

In this study, we have introduced the shifting traveling salesman problem, a transport of a single stack of a finite set of levels such that a number of products must be removed in order to reach products below them in which shifting costs are taken into account. We have formally introduced this problem as a new variant of the traveling salesman problem that is combined with the shifting problem. Our problem addresses the challenge of determining the vehicle's optimal tour that takes account of the products additional movements caused by LIFO policy of stack. We have proposed a suitable mathematical model as a mixed nonlinear program and then we adapted the ant colony and the parallel-ant colony algorithms to solve it which were tested on a number of problem instances of varying problem characteristics from the TSPLIB benchmark sets. Therefore, we have shown that the parallel version of the ant colony algorithm is very efficient in terms of CPU time.

REFERENCES

Agerschou, H., 2004. Planning and Design of Ports and Marine Terminals. 2nd Edn., Thomas Telford Ltd, London, England, UK., ISBN:9780727732248, Pages: 424.

Akyuz, H.M. and C.Y. Lee, 2014. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. Nav. Res. Logistics NRL., 61: 101-118.

Aslidis, A.H., 1989. Combinatorial algorithms for stacking problems. Ph.D Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Avriel, M., M. Penn and N. Shpirer, 2000. Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discrete Appl. Math., 103: 271-279.

Avriel, M., M. Penn, N. Shpirer and S. Witteboon, 1998. Stowage planning for container ships to reduce the number of shifts. Ann. Oper. Res., 76: 55-71.

Battarra, M., G. Erdogan, G. Laporte and D. Vigo, 2010. The traveling salesman problem with pickups, deliveries and handling costs. Transp. Sci., 44: 383-399.

Bektas, T., 2006. The multiple traveling salesman problem: An overview of formulations and solution procedures. Omega, 34: 209-219.

Bigras, L.P., M. Gamache and G. Savard, 2008. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optim., 5: 685-699.

Carrabs, F., J.F. Cordeau and G. Laporte, 2007. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS J. Comput., 19: 618-632.

Casey, B. and E. Kozan, 2012. Optimising container storage processes at multimodal terminals. J. Oper. Res. Soc., 63: 1126-1142.

Cherkesly, M., G. Desaulniers and G. Laporte, 2014. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. Transp. Sci., 49: 752-766.

Cherkesly, M., G. Desaulniers, S. Irnich and G. Laporte, 2016. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks. Eur. J. Oper. Res., 250: 782-793.

Christiansen, M., K. Fagerholt and D. Ronen, 2004. Ship routing and scheduling: Status and perspectives. Trans. Sci., 38: 1-18.

Cordeau, J.F., G. Laporte, P. Legato and L. Moccia, 2005. Models and tabu search heuristics for the berth-allocation problem. Transp. Sci., 39: 526-538.

Cordeau, J.F., M. Iori, G. Laporte and S.J.J. Gonzalez, 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks, 55: 46-59.

Crainic, T.G. and K.H. Kim, 2007. Intermodal transportation. Handbooks Oper Res. Manage. Sci., 14: 467-537.

Dekker, R., P. Voogd and V.E. Asperen, 2006. Advanced methods for container stacking. OR. Spectr., 28: 563-586.

Delgado, A., R.M. Jensen and C. Schulte, 2009. Generating Optimal Stowage Plans for Container Vessel Bays. In: Principles and Practice of Constraint Programming, Gent, I.P. (Ed.). Springer, Berlin, Germany, ISBN:978-3-642-04243-0, pp: 6-20.

Douma, A., M. Schutten andd P. Schuur, 2009. Waiting profiles: An efficient protocol for enabling distributed planning of container barge rotations along terminals in the port of Rotterdam. Transp. Res. Part C. Emerging Technol., 17: 133-148.

Dubrovsky, O., G. Levitin and M. Penn, 2002. A genetic algorithm with a compact solution encoding for the container ship stowage problem. J. Heuristics, 8: 585-599.

Erdogan, G., J.F. Cordeau and G. Laporte, 2009. The pickup and delivery traveling salesman problem with first-in-first-out loading. Comput. Oper. Res., 36: 1800-1808.

Erdogan, G., M. Battarra, G. Laporte and D. Vigo, 2012. Metaheuristics for the traveling salesman problem with pickups, deliveries and handling costs. Comput. Oper. Res., 39: 1074-1086.

Garey, M.R. and D.S. Johnson, 1979. Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York, USA., Pages: 338.

Gunther, H.O. and K.H. Kim, 2006. Container terminals and terminal operations. OR. Spectr., 28: 437-445.

Ichoua, S., M. Gendreau and J.Y. Potvin, 2003. Vehicle dispatching with time-dependent travel times. Eur. J. Oper. Res., 144: 379-396.

Iori, M. and S. Martello, 2010. Routing problems with loading constraints. Top, 18: 4-27.

Petersen, H.L. and O.B. Madsen, 2009. The double travelling salesman problem with multiple stacks formulation and heuristic solution approaches. Eur. J. Oper. Res., 198: 139-147.

Stahlbock, R. and S. Voß, 2008. Operations research at container terminals: A literature update. OR Spectrum, 30: 1-52.

Steenken, D., S. Voß and R. Stahlbock, 2004. Container terminal operation and operations research-a classification and literature review. OR Spectrum, 26: 3-49.

Tagmouti, M., M. Gendreau and J.Y. Potvin, 2007. Arc routing problems with time-dependent service costs. Eur. J. Oper. Res., 181: 30-39.

Tagmouti, M., M. Gendreau and J.Y. Potvin, 2011. A dynamic capacitated arc routing problem with time-dependent service costs. Transp. Res. Part C. Emerging Technol., 19: 20-28.

Veenstra, M., K.J. Roodbergen, I.F.A. Vis and L.C. Coelho, 2017b. The pickup and delivery traveling salesman problem with handling costs. Eur. J. Oper. Res., 257: 118-132.

Veenstra, M., M. Cherkesly, G. Desaulniers and G. Laporte, 2017a. The pickup and delivery problem with time windows and handling operations. Comput. Oper. Res., 77: 127-140.

Vis, I.F.A. and R. de Koster, 2003. Transshipment of containers at a container terminal: An overview. Eur. J. Oper. Res., 147: 1-16.

Wang, N., Z. Zhang and A. Lim, 2014. The Stowage Stack Minimization Problem with Zero Rehandle Constraint. In: Modern Advances in Applied Intelligence, Ali M., J.S. Pan, S.M. Chen and M.F. Horng (Eds.). Springer, Cham Municipality, Switzerland, ISBN:978-3-319-07466-5, pp: 456-465.

Wiel, V.R.J. and N.V. Sahinidis, 1996. An exact solution approach for the time-dependent traveling-salesman problem. Nav. Res. Logist. NRL., 43: 797-820.

Wilson, I.D. and P.A. Roach, 1999. Principles of combinatorial optimization applied to container-ship stowage planning. J. Heuristics, 5: 403-418.

Wilson, I.D. and P.A. Roach, 2000. Container stowage planning: A methodology for generating computerised solutions. J. Operational Res. Soc., 51: 1248-1255.

Wilson, I.D., P.A. Roach and J.A. Ware, 2001. Container stowage pre-planning: Using search to generate solutions, a case study. Knowledge-Based Syst., 14: 137-145.

Yaagoubi, A.E., A.E.H. Alaoui and J. Boukachour, 2016. Multiobjective container barge transport problem. Proceedings of the 2016 3rd International Conference on Logistics Operations Management (GOL), May 23-25, 2016, IEEE, Fez, Morocco, ISBN:978-1-4673-8571-8, pp: 1-6.