CONTAINER MOVEMENT BY TRUCKS IN METROPOLITAN NETWORKS: MODELING AND OPTIMIZATION

Hossein Jula*, Maged Dessouky**, Petros Ioannou***, and Anastasios Chassiakos***

* Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2562
** Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089-0193
*** College of Engineering, California State University at Long Beach, Long Beach, CA 90840-5602

♣ Corresponding author: Email: ioannou@rcf.usc.edu. Tel: (213) 740 4452

Abstract: Today, in the trucking industry, dispatchers still perform the tasks of cargo assignment, and driver scheduling. The growing number of containers processed at marine centers and the increasing traffic congestion in metropolitan areas adjacent to marine ports, necessitates the investigation of more efficient and reliable ways to handle the increasing cargo traffic. In this paper, it is shown that the problem of container movement by trucks can be modeled as an asymmetric “multi-Traveling Salesmen Problems with Time Windows” (m-TSPTW). A two-phase exact algorithm based on dynamic programming (DP) is proposed that finds the best routes for a fleet of trucks. Since the m-TSPTW problem is NP-hard, the computational time for optimally solving large size problems becomes prohibitive. For the case of medium to large size problems, we develop a hybrid methodology consisting of DP in conjunction with genetic algorithms. Computational results demonstrate the efficiency of the exact and the hybrid algorithms.

Keywords: Modelling systems, Travelling salesman, Dynamic programming, Genetic Algorithms, Routing.

---

1 This work is supported by METRANS located at University of Southern California and The California State University at Long Beach, and by the National Science Foundation under grant DMI-9732878. The contents of this paper reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein.
I. INTRODUCTION

The growth in the number of containers has already introduced congestion and threatened the accessibility to many terminals at port facilities [24]. The congestion at a port, in turn, magnifies the congestion in the adjacent metropolitan traffic network and affects the trucking industry on three major service dimensions: travel time, reliability, and cost. Trucking is a commercial activity, and trucking operations are driven by the need to satisfy customer demands and the need to operate at the lowest possible cost [18].

Today, in the trucking industry, human operators (dispatchers) still play the major role in cargo assignment, route planning and driver scheduling. Dispatchers inform drivers about traffic conditions, in addition to assisting them in departure/arrival decisions and providing navigational information [19]. Dispatchers currently obtain information about traffic conditions, mostly through radio traffic reports and through information relayed back by the drivers [11]. The growing number of containers at marine centers and the increasing traffic congestion in metropolitan areas necessitates the investigation of more efficient, reliable and systematic ways to handle the increasing amount of cargo in a metropolitan traffic network.

The purpose of this paper is to investigate methods for improving the scheduling of trucks, where ISO containers\(^2\) need to be transferred between marine terminals, intermodal facilities, and end customers. The objective is to reduce empty miles, and to improve customer service. As a consequence of reduced miles and better service, container terminals can become more competitive, vehicle emissions will be reduced, and drivers will incur less congestion related delays.

\(^2\) Most containers are sized according to International Standards Organization (ISO). Based on ISO, containers are described in terms of TEU (Twenty-foot Equivalent Units) in order to facilitate comparison of one container system with another. A TEU is 8 feet wide, 8 feet high and 20 feet long container. An FEU is an eight-foot high forty-foot container and is equivalent to two TEUs.
In this paper, we show that the container movements by trucks in metropolitan areas can be modeled as a multi-Traveling Salesmen Problem with Time Windows (m-TSPTW). This problem is often referred to as the full-truck-load problem in the academic literature [22]. The problem entails the determination of routes for the fleet of trucks so that the total distribution costs are minimized while various requirements (constraints) are met. The m-TSPTW is an interesting special case of the Vehicle Routing Problem with Time Windows (VRPTW) where the capacity constraints are relaxed. Savelsbergh [21] has shown that finding a feasible solution to the single Traveling Salesman Problem with Time Windows (TSPTW) is an NP-complete problem.

Although there has been a significant amount of research on the VRPTW (e.g., see [2, 4, 5, 7, 10, 14]), there has been little work on the m-TSPTW. Since the m-TSPTW is a relaxation of the VRPTW, it may appear at first that the procedures developed for the latter could be applied to the m-TSPTW. However, as Dumas et al. [6] point out, these procedures are not well suited to the m-TSPTW. Hence, new procedures need to be developed for the m-TSPTW. We note that Dumas et al. [6] presented an efficient exact solution procedure for the single vehicle TSPTW. However, in contrast to the simple transformation of the m-TSP to the TSP, m-TSPTW cannot be easily converted to a single vehicle TSPTW.

The main contributions of this paper are as follows: 1) we show that the container movements by trucks can be modeled as a m-TSPTW problem, and 2) we propose the following two methodologies for solving the m-TSPTW problem.

a. An exact method based on Dynamic Programming (DP) is proposed. The method consists of two phases: 1) generating feasible solutions, and 2) finding the optimum solution among all feasible solutions (set-covering problem). Computational experiments show that the proposed exact method can optimally solve problems with up to 15-20 nodes on randomly generated problems.
b. For medium size problems, we develop a hybrid methodology consisting of dynamic programming for generating feasible solutions in conjunction with genetic algorithms (GA). The GA algorithm is used to find a ‘good’ solution among all feasible solutions. Experimental results show the efficiency of the GA set-covering algorithm for medium to large size problems.

The paper is organized as follows. In Section II, the container movement and trucking operations in metropolitan areas are described. The problem is formulated as an asymmetric m-TSPTW. In Section III, the existing solution methods for the TSPTW and m-TSPTW are briefly reviewed, and two methodologies for solving the m-TSPTW problem are proposed. Section IV concludes this paper.

II. CONTAINER MOVEMENT: PROBLEM DESCRIPTION AND FORMULATION

In this section, it is shown that the container movements by trucks can be modeled as an asymmetric multi-Traveling Salesmen Problem with Time Windows (m-TSPTW). We start with describing the container movement and trucking operations in metropolitan areas.

II.1. Problem Description

Today, in the trucking industry, human operators (dispatchers) perform the tasks of cargo assigning, route planning and driver scheduling. Each day, the list of containers to be handled during the day is passed to the dispatcher early in the morning. The list contains information about the origin and the destination of containers. The dispatcher assigns a driver to each container based on the availability of the driver and his/her skills.

In today’s container industry, there is a lot of discussion about appointment windows. The appointment systems are being considered as part of a solution for terminal congestion. These systems are becoming more important because of the terminals’ need to 'manage the demand' (the flow of trucks). For instance, the Hanjin terminal at the
Port of Long Beach has just started informing the local trucking companies of the list of containers they want to be picked up on the hoot shift (i.e., 3:00 a.m. - 7:00 a.m.).

The problem of interest can be stated more formally as follows: A set of loads (containers) needs to be moved in a metropolitan (local) area. The local area contains one truck depot (which thereafter will be called depot), as well as many end customers including marine terminals and intermodal facilities. Associated to each load is hard time windows imposed by customers for pickup and delivery at origin and destination points, respectively.

A set of trucks (vehicles) is deployed to move the loads among the customers, and the depot. Each truck can only serve a single load (e.g., one FEU size container) at a time, and initially, all trucks are located at the depot. We assume that each driver may not be at the wheel for more than a certain number of hours (working shift) in each working day and has to drive his truck back to the depot within this time limit.

The objective is to minimize the total cost of providing service to the loads within their specified time constraints.

Let \( L \) be a set of \( n \) cargos (containers) to be transferred in a transportation network \( G \), i.e., \( L = \{ l_1, l_2, ..., l_n \} \); and \( V \) be a set of \( p \) vehicles labeled \( v_m, m=1,2,...,p \) assigned to transfer the containers, i.e., \( V = \{ v_1, v_2, ..., v_p \} \).

We assume that, at any time, a vehicle \( v_m \in V \) can transfer at most one single container, say \( l_i \in L \), and that the information of the origin and the destination of container \( l_i \) is known in advance. We denote by \( O(l_i) \) and \( D(l_i) \) the origin and the destination of container \( l_i \), respectively. Container \( l_i \) must be picked up from its corresponding origin during a specific period of time known as the pickup time window and denoted by

\[ \text{3 FEU: Forty-foot Equivalent Units (See also footnote 2).} \]
[\(a_{O(l_i)}, b_{O(l_i)}\)]. Likewise, container \(l_i\) must be delivered at its corresponding destination during a delivery time window denoted by \([a_{D(l_i)}, b_{D(l_i)}]\).

Let \(K(m)\) be the total number of containers assigned to be transferred by vehicle \(v_m \in V\). Let also \(\delta_{mk} \in L\) be the \(kth\) container assigned to vehicle \(v_m\). The sequence of containers assigned to vehicle \(v_m\) is called a route and is denoted by \(r_m\), i.e. \(r_m = \{\delta_{m1}, \delta_{m2}, ..., \delta_{mk}, ..., \delta_{mK(m)}\}\). Route \(r_m\) is said to be feasible if it satisfies the time window constraints at the origins and the destinations of all assigned containers, and the total time needed for traveling on the route is less than a certain amount of time called the working shift (time) and denoted by \(T\).

Figure 1 shows three routes \((r_1, r_2,\) and \(r_3\) starting from the depot and ending at the same depot. Solid lines, in Figure 1, illustrate the traveling between the origin and the destination when the vehicle is loaded, while dashed lines indicate empty traveling between the destination of the last drop-off and the origin of the next pick-up.

Let's denote by \(f(r_m)\) the cost associated with each route \(m\).

\[
f(r_m) = \sum_{k=1}^{K(m)} c_{O(\delta_{mk})}D(\delta_{mk}) + \sum_{k=0}^{K(m)} c_{D(\delta_{mk})}O(\delta_{mk+1})
\]  

(1)

Where:

- \(c_{O(\delta_{mk})}D(\delta_{mk})\) is the cost of carrying the \(kth\) container from its origin to its destination, and

- \(c_{D(\delta_{mk})}O(\delta_{mk+1})\) is the cost of empty travel between the destination of the \(kth\) container to the origin of the \((k+1)th\) container. The depot is denoted by \(k=0\) and \(k=n+1\).
The objective is to find optimum routes for the $p$ vehicles providing the services to the $n$ containers by traveling between origins and destinations of containers and satisfying the time window constraints such that the completion of handling all containers results in minimizing the total travel cost. The objective function, $J$, can be written as follows:

$$J = \text{Min} \sum_{m=1}^{p} f(r_m)$$

Let's assume that the travel cost for a vehicle, either loaded or empty, is static and deterministic, and the cost associated with transferring a container $l_i \in L$ between its origin and destination, $c_{O(l_i)D(l_i)}$, is independent of the order of transferring the
container by a vehicle. Let's also assume that the fleet of vehicles is homogenous. Therefore, no matter what the assignment and order of handling the $n$ containers are, the costs $c_{O(D)}$ don't affect the cost function in (2) and can be considered to be fixed. That is, the total cost function in (2) is only affected by the cost associated with vehicles' empty traveling between the destinations of the $k$th and $(k+1)$th containers; and the problem of interest is reduced to finding the best feasible assignment and sequencing of $n$ containers to $p$ vehicles such that the total empty travel cost of the vehicles is minimum.

![Figure 2: Each origin-destination pair in Figure 1 can be grouped as a node.](image)

Thus, each origin-destination pair, $O(\delta_{mk})-D(\delta_{mk})$, in Figure 1 can be replaced by a node $OD(\delta_{mk})$ where $\delta_{mk} \in r_m$ and $k=1,...,K(m)$, and the cost between two nodes is equal to the cost of empty travel between the destination of the first node to the origin of the second one (see Figure 2).
The time window at node $OD(\delta_{\text{mk}})$ can be expressed in terms of: 1) time window at its origin, 2) time window at its destination, and 3) the traveling time between the origin and the destination, $t_{O(\delta_{\text{mk}})}D(\delta_{\text{mk}})$. Figure 3 demonstrates a typical relation between these three factors and the time window $[a'_D, b'_D]$, where $[a'_D, b'_D]$ is the time window at the destination shifted back in time by $t_{O(\delta_{\text{mk}})}D(\delta_{\text{mk}})$. For the sake of simplicity, we eliminate all subscripts $\delta_{\text{mk}}$ in Figure 3.

Figure 3: Time window at origin $[a_O, b_O]$, destination $[a_D, b_D]$, and time window at destination shifted back in time $[a'_D, b'_D]$.

Figure 4 presents all possible situations between time windows $[a'_D, b'_D]$ and $[a_O, b_O]$. The dashed areas, in Figure 4, indicate the time window at the origin of node $OD$ during which a vehicle can be loaded and yet meet the time window constraint at the destination. Case IV is infeasible and cannot happen in a real situation.

The problem of interest can now be restated as follows: $p$ vehicles are initially located at the depot. They have to visit nodes $OD(i)$, $i=1,...,n$. The task is to select some (or all) of these vehicles and assign routes to them such that each node is visited exactly once during the time window $[a_i, b_i]$, where $[a_i, b_i]$ is expressed as follows (see Figure 4),

$$a_i = a_{O(i)}$$

$$b_i = \min (b_{O(i)}, b_{D(i)} - t_{O(i)}D(i))$$

(3)
The problem now falls in the class of asymmetric Multi-Traveling Salesmen Problems with Time windows (m-TSPTW). In m-TSPTW, \( m \) salesmen are located in a city (i.e. node: \( n+1 \)) and have to visit \( n \) cities (nodes: 1,..,\( n \)). The task is to select some or all of the salesmen and assign tours to them such that in the collection of all tours together the cost (distance) is minimized and each city is visited exactly once within a specified time window [20]. The problem is asymmetric since the traveling cost between each two nodes \( i \) and \( j \) depends on the direction of the move. Note that,

\[
c_{OD(l_i)OD(l_j)} = c_{D(l_i)O(l_j)} \neq c_{D(l_j)O(l_i)} = c_{OD(l_j)OD(l_i)}. \tag{4}
\]

II.2. M-TSPTW Mathematical Formulation

Let \( G=(ND,A) \) be a graph with node set \( ND=\{o,d,N\} \) and arc set \( A=\{(i,j) | \ i, j \in ND\} \). The nodes \{o\} and \{d\} represent the single depot (origin-depot and destination-depot), and \( N=\{1,2,...,n\} \) is the set of customers. To each arc \((i,j)\in A\), a cost \( c_{ij} \) and a
duration of time $t_{ij}$ are associated representing the cost and the time of traveling between nodes $i$ and $j$, respectively. In addition, to each node $i \in ND$, a service time $s_i$ and a time window $[a_i, b_i]$ are associated. The service time $s_i$ is the duration of time for a vehicle to be served at node $i$, and $a_i$ and $b_i$ are the earliest and latest time to visit node $i$, respectively. An arc $(i, j) \in A$ is feasible iff $a_i + s_i + t_{ij} \leq b_j$. Let $V$ be the set of vehicles $v$. A route in the graph $G$ is defined as assigning a set of nodes $r^v = \{o, w_{v1}, w_{v2}, ..., w_{vk}, d\}$ to vehicle $v$ such that each arc in $r^v$ belongs to $A$, and the time that service begins at node $j \in r^v$ is within the time window of that node. Let’s also define:

$$x^v_{ij} = 1 \text{ if arc } (i, j) \in A \text{ is traveled by vehicle } v \text{ and is in the optimal path}^4. \quad x^v_{ij} = 0 \text{ otherwise,}$$

$T^v_i$ is the time when service begins at node $i$ by vehicle $v$.

The m-TSPTW can be formulated as follows:

---

4 The optimal path is a cycle of all nodes with the smallest possible total cost of arcs.
Constraints (5b) require that only one vehicle visit each node in \( N \). Constraints (5c) ensure that at most \(|V|\) number of vehicles are used. To fix the number of vehicles, the inequality should be replaced by equality. Constraints (5d) guarantee that the number of vehicles leaving node \( j \) is the same as the number of vehicles entering the node. Therefore, constraints (5b)–(5d) together enforce that at most \(|V|\) number of vehicles visit all nodes in \( N \) only once. Constraints (5e) enforce the time feasibility condition on consecutive nodes. Constraints (5f) specify the time window constraints at each node. Constraints (5g) require that each vehicle shall be used less than a certain number of hours per day, and finally constraints (5h) are the binary constraints.

### III. PROPOSED SOLUTION METHODS FOR m-TSPTW

The m-TSPTW is an interesting special case of the Vehicle Routing Problem with Time Windows (VRPTW) where the capacity constraints are relaxed. Consequently, one may think of applying the same solution methods for m-TSPTW by relaxing the capacity constraints.
constraints in the VRPTW. Although the idea of using solution methods on VRPTW in m-TSPTW looks very rational, the experimental results show otherwise. In their work, Dumas et al. [6] state: ‘Even though the TSPTW is a special case of VRPTW, the best known approach to the latter problem [4] is not well suited to solve the TSPTW. This column generation approach would experience extreme degeneracy difficulties in this case.’

In the shed of this light researchers have sought methods tailored for the TSPTW and m-TSPTW. However, despite the importance of m-TSPTW in the trucking industry, research on m-TSPTW has been scant [5].

III.1. Solution Methods for TSPTW and m-TSPTW

Since finding a feasible solution to the TSPTW and m-TSPTW is a NP-complete problem [21], most research has focused on heuristic algorithms. Lin and Kernighan [16] proposed a heuristic algorithm based on a k-interchange concept for the TSPTW. The method involves the replacement of $k$ arcs currently in the solution with $k$ other arcs. Lee [15] developed two heuristics based on the Vehicle Scheduling Problem (VSP) for the m-TSPTW. The VSP algorithms are exact in that they can find the optimal solution to the VSP in polynomial time. However, solutions found by VSP algorithms may be infeasible for the m-TSPTW. Two construction heuristics are developed to assign each customer to a route. Improvement heuristics are then developed to combine the initial routes. Calvo [3] proposed the use of a new heuristic method for the TSPTW based on solving an auxiliary assignment problem. To find better solutions, the algorithm uses two objective functions. When the algorithm gets trapped in a local minimum, it uses the second objective function to widen its neighborhood region. Gendreau et al. [9] developed a generalized insertion heuristic algorithm for TSPTW. The algorithm gradually builds a route by inserting at each step a vertex on to the route, and performing a local optimization. Once a feasible route has been determined, a post optimization algorithm is used to improve the objective function.
Despite the difficulty of the TSPTW, few authors have focused on exact solution approaches. Dumas et al. [6] used Dynamic Programming (DP) enhanced by a variety of elimination tests to solve the single vehicle TSPTW optimally. These tests took advantage of the time window constraints to significantly reduce the number of arcs in the graph to eliminate states. The authors managed to solve problems of up to 200 nodes with small window size, and problems of up to 80 nodes with larger time windows.

As we noted previously, in contrast to the relation between the TSP and the m-TSP (see [20]), the m-TSPTW cannot be transformed into an equivalent TSPTW. In this paper, the following methodologies are developed to extend the earlier work on the single vehicle TSPTW:

- An exact two-phase Dynamic Programming (DP), and
- A hybrid methodology consisting of DP in conjunction with genetic algorithms (GA),

We next describe our approaches for solving the m-TSPTW.

III.2. Proposed Exact Method for m-TSPTW

A two-phase Dynamic Programming (DP) based method is proposed for solving m-TSPTW. The method is an extension of the algorithm used for the TSPTW and proposed in [6]. However, contrary to [6], partial paths, which cannot be extended to other nodes, will not be eliminated, as it may be on the optimum set of solutions. Moreover, at each node tests are conducted to ensure the reachability of the depot according to constraint (5g).

In phase one, of the exact method, a forward DP is used to generate all feasible solutions, which will be called states. To reduce the computational time and the number of states, extensive elimination tests are performed before and during running the DP.
algorithm. The elimination tests take advantage of the time window constraints in (5e), (5f) and (5g) to significantly reduce the number of states. The outcome of the first phase will be sets of feasible solutions. The sets, then, will be fed to another DP algorithm in order to find a set of routes with minimum total cost which covers all nodes. That is, the second DP solves a set-covering problem in order to extract the optimum set of solutions among all sets of solutions. In the followings, we explain the exact method in detail.

III.2.1. Methodology:

**Phase One:** We require that the triangular inequality hold for both the travel costs and the travel times between each two nodes of the graph \( G \) (see Section II for definition of \( G \)). That is, for any \( i,j,k \in ND \), we have \( c_{ij} \leq c_{ik} + c_{jk} \) and \( t_{ij} \leq t_{ik} + t_{jk} \). Let \( i \in S \) be the last visited node by vehicle \( v \) (i.e., on path \( v \)); \( t \) be the time at which service begins at node \( i \); and \( S \subseteq N \) be an unordered set of visited nodes. Associated to each state \((S,i,t)\), defined above, is a cost denoted by \( F(S,i,t) \) and defined as the least cost with minimum spanned time of a path starting at node \( \{o\} \in ND \) passing through every node of \( S \) exactly once and ending at node \( i \). Note that, there are several paths that visit set \( S \) and end at node \( i \). Among them, we choose the one with minimum cost and minimum spanned time (see the state elimination test 2, below). Let also \( fr(S) \in N \) be the first node in set \( S \). The starting time of set \( S \) is denoted by \( t_{fr(S)} \) and is defined as \( t_{fr(S)} = \max(a_o, a_{fr(S)} - s_o - t_{o,fr(S)}) \). This time indicates the earliest possible vehicle dispatching time from the depot such that no waiting time is necessary at node \( fr(S) \).

In order to reduce the computational time, two types of elimination tests are performed: arc elimination, and state elimination.

1) **Arc elimination tests:** The arc elimination tests take advantage of the time window constraints (5e), (5f) and (5g) to significantly reduce the number of states. These tests are performed before and during running the DP algorithm.
a. Arc elimination before running the algorithm. Let $EAT(i,j)$ be the earliest arrival time at node $j$ from node $i$. $EAT(i,j)$ is defined as follows:

$$EAT(i,j) = a_i + s_i + t_{ij}$$ (6)

Let also $BEFORE(j)$ be the set of all nodes that should be visited before node $j$, and is defined as follows:

$$BEFORE(j) = \{k \in ND \mid EAT(j,k) > b_k\}$$ (7)

Nodes which can not be covered by set $BEFORE(j)$ will be added to set $FORBID(j)$. In the arc elimination tests, as soon as the algorithm reaches a node $j$ which cannot be added to set $S$ of state $(S,i,t)$, neither node $j$ nor $FORBID(j)$ will be explored at any other state generated from that state. The Forbidden nodes for set $S$ will be kept in the set $U(S)$.

b. Arc elimination during running the algorithm. Given state $(S,i,t)$, $a_i \leq t \leq b_i$, the time to visit node $j \in N$ after node $i$ is $t+s_i+t_{ij}$. The reachability time of the depot, $\{d\} \in ND$, after visiting node $j$ is denoted by $t'$ and is defined as $t' = max(a_j, t+s_i+t_{ij})+s_j+t_{j,d}$. Node $j$ can be added to set $S$ if all of the following tests are satisfied.

$$t + s_i + t_{ij} \leq b_j$$

$$t' \leq b_d$$ (8)

$$t' - t_{fr(s)} \leq T$$

Note that, if node $j$ can not meet any of the tests in (8), node $j$ as well as all nodes in $FORBID(j)$ will be kept in the set $U(S)$ and will not be explored at any other states generated from $S$. 
2) **State elimination tests.** This set of tests implements the dynamic programming algorithm to reduce the number of states: a) during performing phase one, and b) after finishing this phase.

a. **State elimination during running the algorithm.** Given states \((S,i,t_1)\) and \((S,i,t_2)\), the second state is eliminated if \(t_1 \leq t_2\) and \(F(S,i,t_1) \leq F(S,i,t_2)\).

b. **State elimination after finishing the algorithm.** Given states \((S,d,t_1)\) and \((S,d,t_2)\), the second state is eliminated if \(F(S,d,t_1) \leq F(S,d,t_2)\). Test 2-b reduces the number of states passing to the next phase in order to reduce the computational time in phase II.

**Algorithm:**

**Step 1:** Initialization: level \(l=1\),

Form \(\{S,i,t\}: S=\{o,i\}, t=t_{fr(S)}, F(S,i,t)=c_{o,i}\)

\[U(S) = \text{FORBID}(i) \quad \forall (o,i) \in A,\]

**Step 2:** During: Set level \(l=l+1\)

For all states \(S\) in \(l-1\)

If \(U(S) \cup S \neq N\),

For \(\forall (i,j) \in A, j \notin U(S)\), \(i\) is the last node in \(S\), \(j\) satisfying tests in (8)

Generate new state \(\{S_{ij},t_j\}\)

\[S/{d} \cup \{j\} \rightarrow S_{ij}\]

\[t_j = \max(a_j, t + s_i + t_{ij})\]

\[F(S_{ij},t_j) = F(S,i,t) + c_{ij}\]
Perform test 2-part a

Step 3: Termination: If no node was generated in level $l$

Form $S \cup \{d\}$, $F(S, d, t') = F(S, j, t) + c_{j,d}$ where $j$ is the last node in $S$

Perform test 2-part b, Stop

Otherwise: Go to Step 2

Phase Two: The outcome of the first phase is sets of feasible solutions (routes) with their associated costs. Let’s assume that $R$ routes were generated in Phase I. Let $x_r$ be one if the route $r \in R$ is selected among the optimum routes, and zero otherwise. Associated with route $r$ is the cost $F_r$ which was calculated in the previous phase. Let also $\beta_{ir}$ be equal to one if route $r$ visits node $i \in N$, and zero otherwise. The set-covering problem can be formulated as follows:

\[
\text{Min} \quad \sum_{r=1}^{R} F_r x_r
\]

\[
\text{Subject to} \quad \sum_{r=1}^{R} \beta_{ir} x_r = 1 \quad \forall i \in N \quad (9)
\]

\[
x_r = \{0, 1\}
\]

In order to find the set of routes in (9) which covers all nodes in $G$ exactly once with minimum total cost, the routes generated in Phase I are fed to another DP algorithm. That is, the second DP solves a set-covering problem in order to extract the optimum set of solutions among all sets of solutions. It should be mentioned that the set covering problem has also been proven to be NP-hard [8].
In phase II, the state \((S_t,v)\) is defined as follows: \(S_t\subseteq N\) is an unordered set of visited nodes, and \(v\) is the number of routes (vehicles) forming the state \(S_t\). Assigned to each state is a cost denoted by \(g(S_t,v)\) and is defined as the least total cost of routes forming \(S_t\) (covering every node of \(S_t\)). Given states \((S_t, v_1)\) and \((S_t, v_2)\), the second state is eliminated if \(g(S_t,v_1)<g(S_t,v_2)\). Furthermore, if \(g(S_t,v_1)=g(S_t,v_2)\) the second state is eliminated if \(v_1\leq v_2\).

The outcome of Phase II will be a set of routes covering all nodes in \(G\) exactly once with total minimum cost.

**III.2.2. Computational Experiments:**

The exact method was coded in Matlab 5.3 developed by Math Works, Inc. The experimental tests consisted of a Euclidean plane in which customer coordinates were uniformly distributed between 0 and 7 hours, and travel times and costs equal distances. The coordinates of the depot were generated randomly between 3 and 4 hours using the uniform distribution function. The dimensions of the Euclidian plane and the location of the depot were selected such that every customer in the Euclidian plane can be reached and served by at least a truck, and the truck can go back to depot, within a working day, i.e., \(T=10\) hours.

The 'time to start service' at each node was generated as a uniform random variable between 9:00 a.m. to 5:00 p.m. The time window interval length was generated as a uniform random variable in the interval \([0,w]\), where \(w=0.5, 1, 2, \) and \(3\) hours. The service time at each node was assumed to be a uniform random variable generated between 30 minutes to 2 hours. The time window at the depot is set between 6:00 a.m. till 20:00 p.m. and the service time at the depot is assumed to be zero.

Table 1 presents the experimental results for a graph \(G\) with a different number of nodes (customers), \(N\), and different time window lengths, \(w\). The experiments were tested on an Intel Pentium 4, 1.6 GHZ. In Table 1, each set of customers (row) is built
upon the previous row. For instance, for the number of customers (nodes) equal to 10, we used the same randomly generated customers for N=7 and added 3 newly generated customers. At each row of Table 1, we used the same 'times to start service' at customers’ locations while the time window interval lengths were generated randomly, as described above. Table 1 also demonstrates the number of vehicles deployed at each scenario to achieve the minimum cost.

Table 1: The exact method: computational experiments.

<table>
<thead>
<tr>
<th>No of nodes</th>
<th>w=0.5 hour</th>
<th>w=1 hour</th>
<th>w=2 hours</th>
<th>w=3 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimum Cost</td>
<td>CPU time</td>
<td>No Vehicles</td>
<td>Optimum Cost</td>
</tr>
<tr>
<td>7</td>
<td>15.10</td>
<td>0.27</td>
<td>4</td>
<td>15.43</td>
</tr>
<tr>
<td>10</td>
<td>28.82</td>
<td>0.39</td>
<td>6</td>
<td>28.82</td>
</tr>
<tr>
<td>15</td>
<td>48.49</td>
<td>1.81</td>
<td>10</td>
<td>48.49</td>
</tr>
<tr>
<td>20</td>
<td>66.00</td>
<td>85.52</td>
<td>13</td>
<td>66.00</td>
</tr>
</tbody>
</table>

a) NA: The result couldn’t be obtained.

As shown in Table 1, for the first three sets of tests, where the problem size was relatively small (around 15 nodes), the exact method was able to find the optimal solution for all window sizes.

III.3. Genetic Algorithms for m-TSPTW

The exact method is applied to a few sets of problems as shown in Table 1. Since the problem is NP-hard, the computational time for large size problems is fairly high. What we observed in our computational experiments is the DP algorithm for the set covering

---

5 CPU time: is the time in seconds that has been used by the program to obtain the final result.
problem (phase II) is computationally slow for problems involving around 20 or more nodes.

Although the proposed method is capable of finding the exact solution for small size problems, the algorithm becomes computationally very costly for problems of medium or larger size. Hence, it is desirable to find a mechanism that results in a compromise between the quality of the solution and the computational time needed to obtain that solution. Meta-heuristic methods such as Tabu Search, Simulated Annealing (SA), and Genetic Algorithms (GA) offer such a mechanism that forces the algorithm out of the locally optimal solutions in their search for the globally optimal solution. These methods have been recently considered and applied to Set Covering Problems (SCP) with promising results, for instance see [1, 12, 17, 23].

Since it is computationally prohibitive to find the optimal solution among the feasible set of solutions, we propose using a genetic algorithm for solving the second phase in order to find an approximate solution for large size problems.

III.3.1. Methodology:

First, the sets of the routes found in Phase I are encoded in the form of matrix $\beta$ presented in (9). Matrix $\beta$ is an $n \times m$ matrix, where $n$ is the number of customers in graph $G$ and $m$ is the number of the routes generated in phase I. Each element $\beta_{ij}$ of this matrix has a binary value, i.e., it is either one or zero. The $\beta_{ij}$ is one if route $j$ visits node $i \in N$, and zero otherwise. Without loss of generality, we assume that the columns of $\beta$ are ordered in decreasing order of the number of ones in each column (nodes visited by route $j$), and columns with equal number of ones are ordered in increasing order of cost. Equation (10) illustrates a typical matrix $\beta$ for a graph of 4 nodes where 11 feasible routes were generated in Phase I.
A solution to the set-covering problem is a set of routes that visited all customers exactly once. Using the encoded representation of feasible routes in (10), a solution will be a set of columns in matrix $\beta$, e.g. columns 2 and 8, such that the summation over the rows of these columns generate a unit vector column, i.e., has the value one at every row. We used an $m$-bit binary string, which will be called chromosome thereafter, to represent a solution structure. A value 1 for the $i$th bit in the string implies that column $i$ is in the solution set. Equation (11) shows two typical solutions (chromosomes) obtained from matrix $\beta$ in (10).

$$\text{chrml} = [0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0]$$

$$\text{chrml} = [0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

To each chromosome a cost (fitness function) is associated which determines how 'good' each chromosome is. With binary representation, the fitness function $\psi_j$ of chromosome $j$ can be simply calculated by

$$\psi_j = \sum_{i=1}^{m} F_i \cdot \rho_{ij}$$

where $\rho_{ij}$ is the value of the $i$th bit in the chromosome $j$, and $F_i$ is the cost of traveling on route $i$ calculated in phase I.

The initial population of chromosomes is generated by randomly assigning feasible route $i$ to chromosome $j$. The route $i$ is weighted more if it covers more of the remaining nodes with less total costs. Among all population of chromosomes two are “selected”
for generating a new chromosome (offspring). We used the binary tournament selection method by forming two groups of chromosomes with equal number of chromosomes in each group. The chromosomes are randomly placed in each group. One chromosome from each group with the best fitness function is selected to produce an offspring. 

Crossover and mutation operators are then applied on two selected chromosomes to generate the offspring. Let’s assume that chromosomes $j$ and $k$ have been selected for producing the offspring $l$. By applying the crossover operator, the value of $\rho_{il}$ (the $i$th bit in chromosome $l$) is set equal to $\rho_{ij}$ with probability $p$, which is equal to

$$p = \frac{\psi_j}{\psi_j + \psi_k},$$

(13)

and to $\rho_{ik}$ with probability $(1-p)$. Obviously, if the values of $\rho_{ij}$ and $\rho_{ik}$ are equal, the value of $\rho_{il}$ will be equal to this value, i.e., $\rho_{il} = \rho_{ij} = \rho_{ik}$. By applying the mutation operator, the value of $\rho_{il}$ will be inverted by some small probability $q$. However, the new generated offspring, i.e. $l$th chromosome, may not be feasible. A heuristic method can be used to make the chromosome feasible [1].

The newly generated offspring substitutes one of the chromosomes in the initial population with a fitness function worse than the average fitness of all the population. This procedure will continue till the termination criterion is met, which is a predetermined number of iterations.

The GA algorithm is summarized as follows:

Step 1: Form matrix $\beta$; generate the initial population.

Step 2: Select two chromosomes for generating offspring.

Step 3: Apply the crossover operator.
Step 4: Apply the mutation operator.

Step 5: Make the offspring feasible.

Step 6: Calculate the fitness function associated to the offspring.

Step 7: Substitute one of the chromosome in the initial population with the offspring.

Step 8: Repeat steps 2-8 until the termination criterion is met. The best solution is the one with the minimum fitness function in the population.

It is worth mentioning that it is generally believed that GA is slow and would take time to find a high quality solution [25]. The amount of computational effort required by this algorithm depends on the size of the problem, i.e., the number of the nodes in graph $G$ (number of the rows in matrix $\beta$) and the number of the generated feasible solutions in Phase I (number of the columns in matrix $\beta$).

III.3.2. Computational Experiments:

The hybrid GA method was also coded in Matlab 5.3. Table 2 shows the results of using the hybrid GA method for finding the best solution among all feasible solutions for different number of customers while the time window length, $w$, was set to 2 hours. The top number in each cell of Table 2 is the value of the objective function, whereas the lower number is the computational time to find the best solution in seconds, as supplied by Matlab. The same data sets generated in Subsection III.2.2 were used here to evaluate the efficiency of the GA method.

For each set of nodes, the genetic algorithm was applied 10 times in order to assess the reliability and repeatability of the algorithm. A comparison of Table 1 and Table 2 indicates that the hybrid GA algorithm was able to find the optimum solution for 7, 10 and 15 number of nodes at every trial in a relatively short amount of time. These promising results encouraged us to extend the number of nodes in graph $G$ up to 100.
Thus, we used the same methodology described in Subsection III.2.2 to generate new set of nodes (customers). That is, each set of customers (row) is built upon the previous row. Table 2 also shows the result of using the GA algorithm for finding approximate solutions to the graphs having more than 15 nodes.

Table 2: The hybrid GA method: computational experiments, window size \( w=2 \) hours.

<table>
<thead>
<tr>
<th>No of nodes</th>
<th>Best solution in each of 10 trials using GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>13.69/0.39, 13.69/0.16, 13.69/0.22, 13.69/0.16, 13.69/0.11, 13.69/0.16, 13.69/0.22, 13.69/0.16, 13.69/0.17, 13.69/0.28</td>
</tr>
<tr>
<td>10</td>
<td>23.88/0.28, 23.88/0.38, 23.88/1.87, 23.88/0.22, 23.88/0.6, 23.88/0.22, 23.88/0.28, 23.88/0.27, 23.88/0.5, 23.88/0.39</td>
</tr>
<tr>
<td>15</td>
<td>38.38/0.88, 38.38/0.77, 38.38/0.77, 38.38/0.33, 38.38/0.72, 38.38/1.16, 38.38/1.7, 38.38/0.5, 38.38/0.33, 38.38/0.5</td>
</tr>
<tr>
<td>20</td>
<td>52.38/5.76, 52.38/33.45, 52.38/15.21, 52.38/28.28, 52.38/2.86, 52.38/9.5, 52.38/13.34, 52.38/12.869, 52.38/10.39, 52.38/26.96</td>
</tr>
<tr>
<td>50</td>
<td>179.0/179.1, 179.0/85.62, 179.0/234.6, 174.2/89.7, 180.2/239.2, 186.7/204.4, 173.9/157.15, 175.8/255.9, 180.9/203.5, 181.2/178.4, 182.3/270.6</td>
</tr>
<tr>
<td>100</td>
<td>337.8/1865, 337.0/2062, 339.7/2179, 340.6/1876, 337.1/2079, 340.7/2232, 336.7/1678, 342.1/1806, 338.0/1415, 339.8/1569</td>
</tr>
</tbody>
</table>

a) The best value obtained from GA method / CPU time to find the best solution in seconds.

The GA hybrid algorithm was also tested on graph \( G \), explained above, while the time window interval varies between 0.5 to 3 hours, i.e. \( w=0.5, 1, 2, \) and 3 hours. The results are summarized in Table 3. Each cell in Table 3 was obtained by averaging the results acquired from 10 trials.

We next benchmark the proposed exact and hybrid genetic algorithm methods against a heuristic insertion method similar to that presented in [13]. Table 4 summarizes and compares the cost and the CPU time of the exact method, the hybrid Genetic Algorithm (GA) method, and the insertion method. As is shown in Table 4, the exact method is efficient for relatively small size problems consisting of a few nodes. GA is capable of finding optimum solution for small size problems, and significantly outperforms the insertion heuristic method when the number of nodes is less than 100.
Table 3: The hybrid GA method: the summary of the computational experiments for different window size (based on the results of 10 trials).

<table>
<thead>
<tr>
<th>No of nodes</th>
<th>( w=0.5 ) hour</th>
<th>( w=1 ) hour</th>
<th>( w=2 ) hours</th>
<th>( w=3 ) hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Cost</td>
<td>CPU time</td>
<td>Best Cost</td>
<td>CPU time</td>
</tr>
<tr>
<td>7</td>
<td>15.10°</td>
<td>0.20</td>
<td>15.43°</td>
<td>0.18</td>
</tr>
<tr>
<td>10</td>
<td>28.82°</td>
<td>0.19</td>
<td>28.82°</td>
<td>0.19</td>
</tr>
<tr>
<td>15</td>
<td>48.49°</td>
<td>0.59</td>
<td>48.49°</td>
<td>0.99</td>
</tr>
<tr>
<td>20</td>
<td>66.00°</td>
<td>2.17</td>
<td>66.00°</td>
<td>2.89</td>
</tr>
<tr>
<td>30</td>
<td>112.2</td>
<td>28.76</td>
<td>104.0</td>
<td>23.64</td>
</tr>
<tr>
<td>50</td>
<td>196.9</td>
<td>126.7</td>
<td>189.5</td>
<td>240.9</td>
</tr>
<tr>
<td>100</td>
<td>359.7</td>
<td>1234</td>
<td>352.4</td>
<td>1368</td>
</tr>
</tbody>
</table>

o) Optimum Value.

Table 4: Comparing exact, hybrid GA, and insertion methods, \( N=20 \) nodes, \( w=2 \) hours.

<table>
<thead>
<tr>
<th>No of nodes</th>
<th>Dynamic Programming</th>
<th>Genetic Algorithm (^a)</th>
<th>Insertion method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>CPU time</td>
<td>Cost</td>
</tr>
<tr>
<td>7</td>
<td>13.69</td>
<td>0.38</td>
<td>13.69</td>
</tr>
<tr>
<td>10</td>
<td>23.88</td>
<td>1.70</td>
<td>23.88</td>
</tr>
<tr>
<td>15</td>
<td>35.38</td>
<td>326.4</td>
<td>35.38</td>
</tr>
<tr>
<td>20</td>
<td>NA(^b)</td>
<td>NA</td>
<td>52.44</td>
</tr>
<tr>
<td>30</td>
<td>NA</td>
<td>NA</td>
<td>98.97</td>
</tr>
<tr>
<td>50</td>
<td>NA</td>
<td>NA</td>
<td>179.4</td>
</tr>
<tr>
<td>100</td>
<td>NA</td>
<td>NA</td>
<td>338.9</td>
</tr>
</tbody>
</table>

\(^a\) In average (based on the results of 10 trials).
\(^b\) NA: The result couldn’t be obtained.
It should be noted that in our computational experiments the maximum number of generated solutions (offspring) in GA was limited to 1000. Obviously, by increasing this number a better solution may be found.

IV. CONCLUSIONS

In this paper, we investigated the cargo movement in metropolitan areas adjacent to marine ports. In particular, we were interested in improving the methods for truck scheduling and route planning, where ISO containers need to be transferred between marine terminals, intermodal facilities, and end customers. The objective was to reduce empty miles, and to improve customer service. We showed that the container movement by trucks can be modeled as an asymmetric multi-Traveling Salesmen Problem with Time Windows (m-TSPTW). Moreover, we proposed two methodologies for solving the m-TSPTW:

- An exact two-phase Dynamic Programming (DP), and
- A hybrid methodology consisting of DP in conjunction with genetic algorithms (GA).

The results of our computational experiments indicate that the exact method was efficient for relatively small size problems consisting of a few nodes. However, the hybrid GA was capable of finding the optimum solution for small size problems and a sub-optimum solution for medium to large size problems (more than 30 nodes).

Acknowledgments

We would like to thank Ms. Patty Senecal, Mr. Mike Johnson, and Mr. J.R. Barba of Transport Express for supplying us with useful information on the problem.

References


