

CONTAINER MOVEMENT BY TRUCKS IN METROPOLITAN NETWORKS: MODELING AND OPTIMIZATION¹

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: Container movement by trucks with time constraints at origins and destinations is modeled as an asymmetric “multi-Traveling Salesmen Problem with Time Windows” (m-TSPTW) with social constraints. 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 large size problems, we develop a hybrid methodology consisting of DP in conjunction with genetic algorithms. The developed algorithms are compared with an insertion heuristic method. Computational results demonstrate the efficiency of the developed algorithms.

Keywords: Traveling salesman, Time windows, Dynamic programming, Genetic algorithms, Heuristic, Routing.

I. INTRODUCTION

With all the benefits of containerization come increased operational complexities. Over the period of 1990 to 2000, for instance, the growth in the container traffic in the Los Angeles and Long Beach (LA/LB) twin ports, the U.S. largest ocean freight hub and busiest container port complex, has been quite significant. The average annual growth in the LA/LB ports has been 9.2%, which easily passed

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.

the forecasted annual growth of 6.2% [21, 30]. Even for the lower forecasted rate of 6.2%, the estimated container traffic in the year 2020 will be around 28 million TEU²s or almost 15.1 million containers (at the rate of 1.85 TEU/container³). That is, by the year 2020 the volume of containers that will be moving through the combined LA/LB port will be at least three times the current volume [21].

The growth in the number of containers has already introduced congestion and threatened the accessibility to many terminals at port facilities [31]. In a recent study, Barber and Grober [2] found that 40% of trucks, visiting the LA/LB terminals, are involved in more than two hours waiting time, with almost a quarter of the transactions involving a wait in the range of 2 to 3 hours. In addition to driver inefficiency, the traffic congestion and long queues at the gates of terminals are a main source of air pollution (especially diesel toxins), wasted energy, increasing cost imposed by the volume of trucks on roadway for maintenance, etc. [3].

The congestion at ports, in turn, magnifies the congestion in the adjacent metropolitan traffic networks 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 [22]. Through observing a dispatcher's activity at a local trucking company in the Los Angeles area in the year 2001, we observed that the dispatcher played a major role in container assignment, route planning and driver scheduling. The dispatcher assigned a driver to each container based on the availability of the driver and his skills without concerns of optimizing a cost function. At most two containers were assigned to each driver at a time. The dispatcher assigned two containers if the delivery point of the first one was relatively close to the pick up point of the second one. Otherwise, only one container was assigned to the driver. Upon finishing his job(s), the driver would ask the dispatcher for a new job. If the new job needed to be accompanied by formal documentation or no job was available at the time, the driver would be asked to go back to the depot. Otherwise, the new job would be given to him/her via the cell phone.

² Most containers are sized according to International Standards Organization (ISO). Based on ISO, containers are described in terms of TEU (Twenty-foot Equivalent Unit) 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 long container and is equivalent to two TEUs.

³ Typically a container is either 20 ft or 40 ft in length, but the vast majority of containers are forty feet. It is estimated that on the average, a random container would correspond to 1.85 TEU [30].

During the day of our visit, about 65 containers were moved among depot, end customers, local intermodal facilities and the Los Angeles and Long Beach container terminals. We were told that this number of containers is fairly typical for a day. The company blamed the drivers' inefficiencies to the long queues at the terminals' gates. The company was hoping that the newly initiated appointment window system (hereafter time-window system) could manage the flow of trucks at the gates of the terminals [28].

The time-window system requires that freight carriers deliver/pick-up their cargo within a specified time period. The system arises naturally in the trucking industry due to the commitments made to the end customers for just-in-time (JIT) cargo delivery and pickup, and due to the limited availability of certain resources and increasing congestion at container terminals. It is expected that through implementing a time-window system, container terminals could become more competitive, vehicle emissions would be reduced, and drivers would incur less congestion related delays.

In this paper, the container movement by trucks with time constraints at both origins and destinations is modeled as an asymmetric "multi-Traveling Salesmen Problems with Time Windows" (m-TSPTW). This problem is often referred to as the full-truck-load problem [26]. 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 [25] has shown that even 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 [5, 7, 8, 10, 14, 17]), little work has been done 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. [9] pointed out, these procedures are not well suited even for a single vehicle TSPTW.

Since finding feasible solutions to the TSPTW and m-TSPTW are NP-hard problems, most research has focused on heuristic algorithms [6, 13, 18, 19, 32]. These approaches are reviewed later in this paper. Noteworthy, very few authors have focused on exact solution approaches [1, 9, 11]. Dumas *et al.* [9] used Dynamic Programming (DP) enhanced by a variety of elimination tests to

optimally solve the single vehicle TSPTW. These tests took advantage of the time window constraints to significantly reduce the number of arcs in the graph to eliminate the 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. Focacci et al. [11] proposed a hybrid exact algorithm for solving the TSPTW based on a constraint programming framework, which finds feasible paths in conjunction with a propagation algorithm to find the optimum solution. The proposed algorithm managed to optimally solve asymmetric TSPTW problems with up to 70 nodes

It should be noted that in contrast to the relation between the Traveling Salesman Problem (TSP) and the multi-traveling salesmen problem (m-TSP) [24], the m-TSPTW cannot be converted into an equivalent TSPTW in general. The transformation of an instance of m-TSP to the TSP problem embraces the fact that the TSP and m-TSP are both problems in one dimension, *space*. The TSPTW and m-TSPTW, however, are problems in two dimensions: *space* and *time*. Any solution to TSPTW and m-TSPTW should satisfy both space and time constraints. An instance of m-TSPTW may be feasible in time while any polynomial time transformation of that instance to an instance of the single vehicle TSPTW would be infeasible. In other words, in general, such a transformation does not exist.

The purpose of this paper is to investigate methods for improving the scheduling of trucks, where ISO containers need to be transferred between marine terminals, intermodal facilities, and end customers. Each of these customers/facilities may have imposed time-window constraints on pick-up/drop-off containers. The objective is to reduce empty miles, and to improve customer service (i.e., satisfying the time-windows at customers/facilities locations).

The main contributions of this paper are as follows:

- 1) We show that container movement by trucks in metropolitan networks with time constraints at both origins and destinations can be modeled as an asymmetric m-TSPTW problem with additional social constraints. The social constraints are enforced by trucking companies to ensure that drivers return to their depots before a certain amount of time.

- 2) We propose the following two methodologies for solving the container movement 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. A hybrid methodology consisting of dynamic programming for generating feasible solutions in conjunction with genetic algorithms (GA) is developed. The GA algorithm is used to find a ‘good’ solution among all feasible solutions. Experimental results show the efficiency of the hybrid GA algorithm in solving problems with up to 100 nodes.

Finally, we benchmark the proposed exact and hybrid genetic algorithm methods against a heuristic insertion method similar to that presented in [16]. The method involves inserting nodes into the routes sequentially. Experimental results show that the heuristic insertion method is computationally very fast and it is fairly efficient for medium to large size problems (compared to the real-life problem discussed earlier).

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 three methodologies for solving the m-TSPTW problem with additional social constraints are developed and compared. Section IV concludes this paper.

II. CONTAINER MOVEMENT: PROBLEM DESCRIPTION AND FORMULATION

The container movement problem in metropolitan areas can be stated as follows: A set of containers needs to be moved in a metropolitan (local) area by a local trucking company. A set of trucks, initially located at the company’s depot (which hereafter will be called depot), is deployed to move these containers among the depot, end customers, and service stations (STs). Service stations include marine terminals and intermodal facilities. Associated with each container is a time window imposed by customers and STs for pickup and/or delivery at origin and destination points.

We assume that each truck can only serve a single load (e.g., one FEU size container) at a time, and that each driver may not be at the wheel for more than a certain number of hours (working shift) during a day. In other words, each driver has to drive his truck back to the depot before his shift comes to end. This time limit is a social constraint enforced by trucking companies to conform to

local or federal laws. Considering a relatively small transportation network (metropolitan network), we assume that a driver can serve even the farthest container in the network within his working shift.

Assuming that the trucking company knows all its daily loads to be delivered/picked-up a priori, the objective is to minimize the total cost of providing service to the customers and STs within their specified time constraints. In other words, the objective is to select some or all of the trucks and assign a set of containers to each one of them such that in the collection all containers are served exactly once within their time constraints and the total cost (distance) is minimized.

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

We assume that, at any time, a vehicle $v_m \in V$ can transfer at most a 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 $[a_{O(l_i)}, b_{O(l_i)}]$. Likewise, container l_i must be delivered at its corresponding destination during a delivery time window denoted by $[a_{D(l_i)}, b_{D(l_i)}]$. With a hard time window $[a_k, b_k]$ associated with a container at a customer location, we assume that if a vehicle arrives at that location at any time $t < a_k$, it waits till the customer is ready to service it, i.e., till time $t \geq a_k$. The vehicle cannot be served if it arrives at the location at any time $t > b_k$.

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 k th 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., $r_m = \{\delta_{m1}, \delta_{m2}, \dots, \delta_{mk}, \dots, \delta_{mK(m)}\}$. Route r_m is said to be feasible if it satisfies the time window constraints at the origins and 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 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 $f(r_m)$ denote the cost associated with route m .

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

where $c_{O(\delta_{mk})D(\delta_{mk})}$ is the cost of carrying the k th 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 k th container to the origin of the $(k+1)$ th container. The cost $c_{D(\delta_{mK(m)})O(\delta_{m(K(m)+1)})}$ is the cost of returning the empty trucks back to the depot at the end of their routes. The depot is denoted by $k=0$.

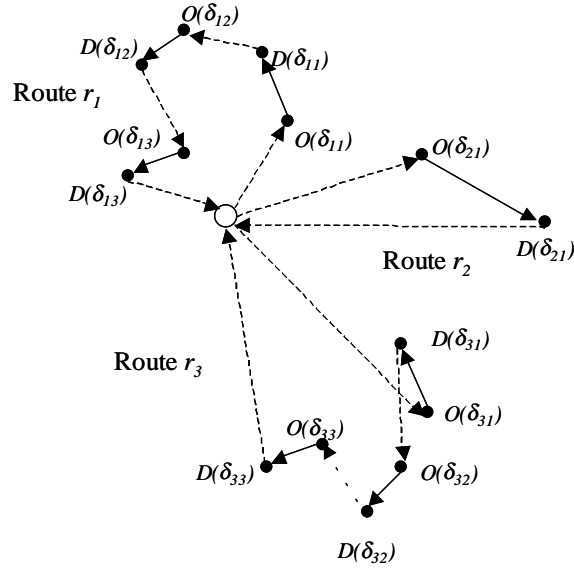


Figure 1: Typical routes starting from depot and ending at the same depot. The large empty circle denotes the depot. Each small black circle denotes the origin (O) or the destination (D) of a container.

In (1), for the sake of simplicity, we assume that no cost is associated to the loading/unloading operations at the customers and STs' locations.

The objective is to find optimum routes for the p vehicles providing the services to the n containers by traveling between the origins and destinations of the 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:

$$J = \text{Min} \sum_{m=1}^p f(r_m) \quad (2)$$

where r_m denotes the route m , $f(r_m)$ is the cost associated with route m , and p is the number of routes (vehicles).

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 $c_{O(\cdot)D(\cdot)}$ don't affect the cost function in (2) and are 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. 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.

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 $k=1, \dots, K(m)$. Figure 2 illustrate the modified routes where the pairs origin-destination, $OD(\delta_{mk})$, are treated as nodes.

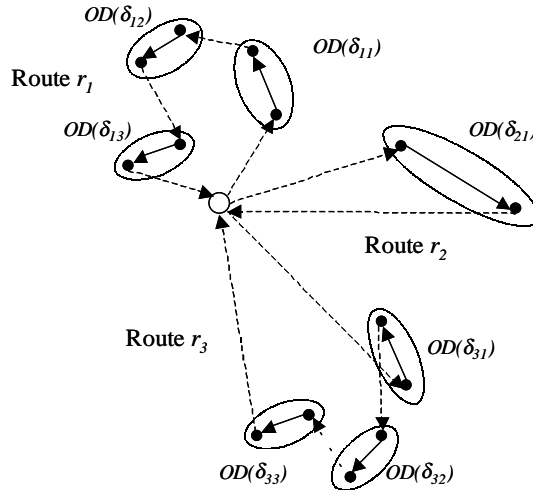


Figure 2: Each origin-destination pair in Figure 1 can be grouped as a node.

Given the parameters (e.g., traveling costs on links, time windows at nodes, etc.) of the original network in Figure 1, the parameters of the modified network in Figure 2 is determined as follows:

The *cost* of traveling between each two nodes $OD(\delta_{mk})$ and $OD(\delta_{mk+1})$ in Figure 2 is equal to the cost of empty traveling between the destination of the first node, $D(\delta_{mk})$, to the origin of the second one, $O(\delta_{m(k+1)})$. The *service time* at each node $OD(\delta_{mk})$ is defined as the time required for carrying a load between its origin and destination points (i.e., between $O(\delta_{mk})$ and $D(\delta_{mk})$) together with the time needed for loading a container and unloading it at these points, respectively. The service time includes the waiting time at the destination, if any. The *time window* to visit each node $OD(\delta_{mk})$ can be expressed in terms of: 1) time window at its origin $[a_{O(\delta_{mk})}, b_{O(\delta_{mk})}]$, 2) time window at destination $[a_{D(\delta_{mk})}, b_{D(\delta_{mk})}]$, and 3) the traveling time between the origin and the destination, $t_{O(\delta_{mk})D(\delta_{mk})}$. Figure 3 demonstrates a typical relation between these three elements. For the sake of simplicity, we eliminate all subscripts δ_{mk} in Figure 3. The time window denoted by $[a'_D, b'_D]$ in Figure 3 is the time window at the destination shifted back in time by $t_{O(\delta_{mk})D(\delta_{mk})}$.

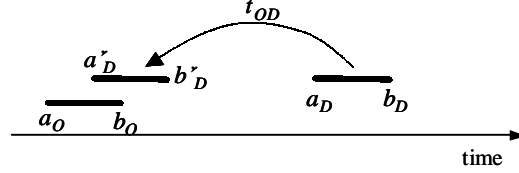


Figure 3: A typical relation between time window at origin $[a_O, b_O]$, destination $[a_D, b_D]$, and time window at destination shifted back in time $[a'_D, b'_D]$.

Figure 4 presents all possible relations 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. For instance, Case I in Figure 4 indicates that if a vehicle visited node OD at any time in the interval $[a_O, b_O]$, it would be served at both the origin and destination of this node. Case I also indicates that if a loaded vehicle leaves the origin O at any time $\tau \in [a_O, a'_D)$, it will reach the destination D , prior to time a_D . Therefore, the vehicle has to wait at the destination D for a period of time equal to $a'_D - \tau = a_D - t_{OD} - \tau$. If the vehicle leaves the origin O at any time $\tau \in [a'_D, b_O]$, it will be unloaded at D without any delay. Moreover, if the vehicle reaches the origin O at time $\tau \in (b_O, b'_D]$, it cannot

be loaded at the origin even though the vehicle can meet the time constraint (i.e., can be unloaded) at the destination.

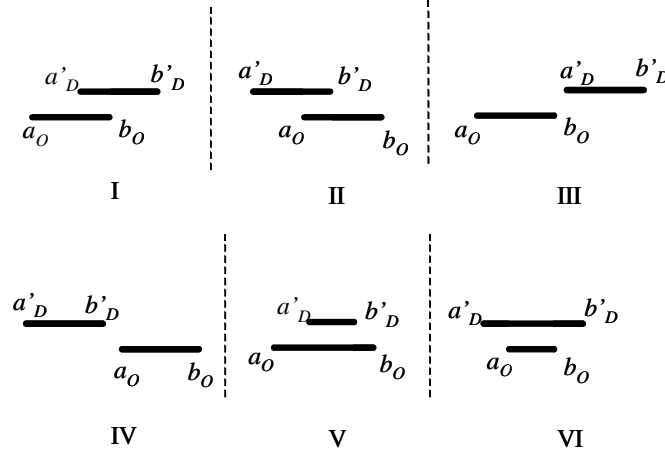


Figure 4: All possible relations between time window at origin, and time window at destination shifted back in time. The dashed area presents the time window at node OD during which a vehicle can be loaded.

Having determined the parameters of the modified network in Figure 2, we can now restate the problem of route planning for a fleet of trucks as follows: p vehicles are initially located at the depot. They have to visit nodes $OD(l_i)$, $i=1,\dots,n$. 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).

$$\begin{aligned}
 a_{l_i} &= a_{O(l_i)} \\
 b_{l_i} &= \min(b_{O(l_i)}, b_{D(l_i)} - t_{O(l_i)D(l_i)})
 \end{aligned}
 \tag{3}$$

Note that the possibility of $b_{l_i} < a_{l_i}$ in (3) corresponds to the Case IV in Figure 4 which is an infeasible case and may not occur in a real situation.

The problem now falls in the class of asymmetric Multi-Traveling Salesmen Problems with Time windows (m-TSPTW). In the m-TSPTW, m salesmen are located in a city (i.e. node: $n+1$) and have to visit n cities (nodes: $1,\dots,n$). 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 (e.g., distance) is minimized and each city is visited exactly once within a specified time window [24]. The problem is asymmetric since the

traveling cost between each two nodes i and j depends on the direction of the move. Note that, since $c_{D(i)O(l_j)} \neq c_{D(l_j)O(i)}$ we have

$$c_{OD(l_i)OD(l_j)} \neq c_{OD(l_j)OD(l_i)} \quad (4)$$

As discussed earlier, we also require that the total time needed for a vehicle to visit all nodes on its route and return to the depot be less than a certain amount of time, working shift (time) T .

II.1. 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 $N=\{1,2,\dots,n\}$ 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 $r^v = \{o, w_1^v, w_2^v, \dots, w_k^v, d\}$ 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:

$x_{ij}^v = 1$ if arc $(i,j) \in A$ is traveled by vehicle v and is in the optimal path⁴. $x_{ij}^v = 0$ otherwise,

T_i^v 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.

$$\text{Min} \quad \sum_{v \in V} \sum_{(i,j) \in A} c_{ij} x_{ij}^v \quad (5)$$

$$\text{Subject to} \quad \sum_{v \in V} \sum_{j \in N \cup \{d\}} x_{ij}^v = 1 \quad \forall i \in N \quad (6.a)$$

$$\sum_{v \in V} \sum_{j \in N} x_{oj}^v \leq |V| \quad \forall i \in N, v \in V \quad (6.b)$$

$$\sum_{j \in N \cup \{d\}} x_{ij}^v - \sum_{j \in N \cup \{o\}} x_{ji}^v = 0 \quad \forall i, j \in ND, v \in V \quad (6.c)$$

$$x_{ij}^v (T_i^v + s_i + t_{ij} - T_j^v) \leq 0 \quad \forall i, j \in ND, v \in V \quad (6.d)$$

$$a_i \leq T_i^v \leq b_i \quad \forall i \in ND, v \in V \quad (6.e)$$

$$x_{ij}^v \in \{0, 1\} \quad \forall (i,j) \in A, v \in V \quad (6.f)$$

Constraints (6.a) require that only one vehicle visit each node in N . Constraints (6.b) ensure that at most $|V|$ number of vehicles are used. To fix the number of vehicles, the inequality should be replaced by equality. Constraints (6.c) guarantee that the number of vehicles leaving node j is the same as the number of vehicles entering the node. Therefore, constraints (6.a) - (6.c) together enforce that at most $|V|$ number of vehicles visit all nodes in N only once. Constraints (6.d) enforce the time feasibility condition on consecutive nodes. Constraints (6.e) specify the time window constraints at each node. Finally constraints (6.f) are the binary constraints.

In addition to constraints (6.a) to (6.f), we also require the following constraint be met in order to implement the social constraints imposed on truck drivers.

$$T_d^v - T_o^v \leq T \quad \forall v \in V \quad (6.g)$$

Constraints (6.g) require that each driver shall not be on the wheel more than a certain number of hours, T , in each single day. Note that by adding (6.g) to the m-TSPTW model, we assume that truck drivers have to return their trucks back to depot before the end of their shifts.

III. PROPOSED SOLUTION METHODS FOR M-TSPTW

As discussed earlier, 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 the m-TSPTW by relaxing the capacity 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.* [9] state: '*Even though the TSPTW is a special case of VRPTW, the best known approach to the latter problem [7] is not well suited to solve the TSPTW. This column generation approach would experience extreme degeneracy difficulties in this case.*'

In response to these difficulties 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 [8].

Lin and Kernighan [19] 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 [18] 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 [6] 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.* [13] developed a generalized insertion heuristic algorithm for the 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. Wang and Regan [32] described an iterative solution technique for the m-TSPTW, which extensively uses a time-window discretization scheme to partition the time windows.

In the following sections, three methodologies are studied to extend the earlier works to the m-TSPTW problem with additional constraints (6.g). These methodologies are

- a. An exact method based on Dynamic Programming (DP),
- b. A hybrid methodology consisting of dynamic programming in conjunction with genetic algorithms (GA), and

- c. An insertion heuristic method.

III.1. Proposed Exact Method for the m-TSPTW with Additional Social Constraints

A two-phase Dynamic Programming (DP) based method is proposed for the m-TSPTW with additional social constraints. The method is an extension of the algorithm used for the TSPTW and proposed in [9]. Our approach, however, differs from [9] in two dimensions. First, since our problem involves multiple vehicles, contrary to [9], partial paths which cannot be extended to other nodes will not be eliminated. These paths might be in the optimal solution set. Second, our problem of interest contains additional social constraints. Therefore, we keep track of the amount of time a truck has been moving containers out of the depot. To do so, at each node, tests are conducted to ensure the reachability of the depot according to constraints (6.g).

In phase one, 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, elimination tests are performed before and during running the DP algorithm. The elimination tests take advantage of the time window constraints in (6.d), (6.e) and (6.g) to 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, in Phase 2 the second DP solves a set-covering problem in order to extract the optimum set of solutions among all sets of solutions. In the following, we explain the exact method in detail.

III.1.1. Methodology:

Phase One: Let's consider the graph G as described earlier in Section II.1. We require that the triangular inequality hold for both the travel costs and travel times between each two nodes of the graph G . That is, for any $i, j, k \in ND$, we have $c_{ik} \leq c_{ij} + c_{jk}$ and $t_{ik} \leq t_{ij} + t_{jk}$. Let $S \subseteq ND$ be an unordered set of visited nodes by a vehicle v (the path taken by the vehicle), $U \subset N$ be the set of nodes which cannot be added to set S , $i \in S$ be the last visited node; and t_i be the time at which service begins at node i on path S . The four-tuple (S, U, i, t_i) defines a state at which a vehicle may reside. Associated to each state, is a cost denoted by $F(S, U, i, t_i)$ defined as the least cost with minimum spanned time of a path starting at the origin, $\{o\}$, 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 $fr(S) \in N$ be the first node in set S . The starting time of set S , denoted by $t_{fr(S)}$, is defined as

$$t_{fr(S)} = \max(a_o, a_{fr(S)} - s_o - t_{o,fr(S)}) \quad (7)$$

where a_o and $a_{fr(S)}$ are the earliest times to serve the vehicles at the origin and node $fr(S)$, respectively, s_o is the service time at the origin, and $t_{o,fr(S)}$ is the traveling time from the origin to node $fr(S)$. The time $t_{fr(S)}$ indicates the earliest possible time a vehicle may start its path from origin $\{o\}$ without having to wait at node $fr(S)$.

The Dynamic Programming (DP) algorithm in Phase One starts from the origin. At each step, it adds all feasible, uncovered nodes to the paths. When a node is added to path S , the set U is updated. The nodes in U will not be explored at any generated path from S . This procedure continues till all nodes are covered and all feasible paths are generated. In order to reduce the computational time, two types of elimination tests are performed: arc and state elimination tests.

1) *Arc elimination tests*: The arc elimination tests take advantage of the time window constraints (6. d), (6.e) and (6.g) 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. For all $(i,j) \in A$, the earliest possible arrival time at node j from node i is denoted by $E(i,j)$ and is defined as follows [9]:

$$E(i, j) = a_i + s_i + t_{ij} \quad (8)$$

Let $B(j)$ be the set of all nodes that can not be visited after node j for any circumstance. $B(j)$ is defined as follows:

$$B(j) = \{k \in N \mid E(j, k) > b_k\} \quad (9)$$

In arc elimination tests, when the algorithm adds node j to set S of state (S, U, i, t_i) , the nodes in the set $B(j)/S$ will be added to set U .

- b. Arc elimination during running the algorithm. Given state (S, U, i, t) , $a_i \leq t \leq b_i$, and arc $(i, j) \in A$, $j \notin S$, the depot $\{d\} \in ND$ can be reached after visiting node j at the time

$$t'_j = \max(a_j, t_i + s_i + t_{ij}) + s_j + t_{j,d} \quad (10)$$

Node j can be added to set S if all of the following tests are satisfied.

$$\begin{aligned} t + s_i + t_{ij} &\leq b_j \\ t' &\leq b_d \\ t' - t_{fr(S)} &\leq T \end{aligned} \quad (11)$$

Note that, if node j can not meet any of the tests in (11), node j will be added to set U .

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

Algorithm:

Step 1: Initialization: level $l=1$,

Form $\{S, U, i, t\}$: $S = \{o, i\}$, $fr(S) = \{i\}$, $t = t_{fr(S)}$, $F(S, i, t) = c_{o,i}$ and U according to elimination test (1.a),

Step 2: During: Set level $l=l+1$

For all states S in $l-1$

If $U \cup S \neq \{o, N\}$, and j satisfying tests in (11),

For $\forall (i, j) \in A$, $j \notin U$, and i is the last node in S ,

$$t_j = \max(a_j, t_i + s_i + t_{ij})$$

$$U \leftarrow U \cup \{\text{set obtained from test (1.a)}\}$$

Generate new state $\{\{S,j\}, U,j,t_j\}$

$$F(\{S,j\}, U,j,t_j) = F(S, U, i, t_i) + c_{ij},$$

Perform test 2-part a

Step 3: Termination: If no node was added to states in level l

Form $F(\{S,d\}, U,d,t) = F(S, U, j, t) + c_{j,d}$ where j is the last node in S

Perform test 2- part b, Stop

Phase Two: The outcome of the first phase is sets of feasible solutions (routes) in state form (S, U, d, t_d) with associated costs $F(S, U, d, t_d)$. 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 β_{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:

$$\begin{aligned}
 & \text{Min} && \sum_{r=1}^R F_r x_r \\
 & \text{Subject to} && \sum_{r=1}^R \beta_{ir} x_r = 1 && \forall i \in N \\
 & && x_r = \{0,1\}
 \end{aligned} \tag{12}$$

In order to find the set of routes in (12) 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 [12].

In phase II, the state (St, v) is defined as follows: $St \subseteq N$ is an unordered set of visited nodes, and v is the number of routes (vehicles) forming the state St . Assigned to each state is a cost denoted by $g(St, v)$ and is defined as the least total cost of routes forming St (covering every node of St). Given states (St, v_1) and (St, v_2) , the second state is eliminated if $g(St, v_1) < g(St, v_2)$. Furthermore, if $g(St, v_1) = g(St, v_2)$ 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.1.2. 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 with the travel time and cost equal to the distance. 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 the 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 8:00 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.

| No of nodes | w=0.5 hour | | | w=1 hour | | | w=2 hours | | | w=3 hours | | |
|-------------|--------------|-----------------------|-------------|--------------|----------|-------------|-----------------|----------|-------------|--------------|----------|-------------|
| | Optimum Cost | CPU time ^a | No Vehicles | Optimum Cost | CPU time | No Vehicles | Optimum Cost | CPU time | No Vehicles | Optimum Cost | CPU time | No Vehicles |
| 7 | 15.10 | 0.27 | 4 | 15.43 | 0.28 | 4 | 13.69 | 0.38 | 3 | 13.69 | 0.44 | 3 |
| 10 | 28.82 | 0.39 | 6 | 28.82 | 0.39 | 6 | 23.88 | 1.70 | 5 | 23.16 | 3.40 | 4 |
| 15 | 48.49 | 1.81 | 10 | 48.49 | 3.35 | 10 | 35.38 | 326.3 | 7 | 37.70 | 625.7 | 7 |
| 20 | 66.00 | 85.52 | 13 | 66.00 | 302.3 | 13 | NA ^b | NA | NA | NA | NA | NA |

a) CPU time: is the time in seconds that has been used by the program to obtain the final result.

b) 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. Also, as expected, the computational time increases as the time window size increases since fewer states can be eliminated.

III.2. Genetic Algorithms for m-TSPTW with additional social constraints

Although the proposed exact method is capable of finding the optimum solution for small size problems, the algorithm becomes computationally very costly for problems of medium or larger size. What we observed in our computational experiments is that the DP algorithm for the set covering problem (phase II) is computationally slow for problems involving around 20 or more nodes.

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 a local minimum solution in their search for the global optimal solution. These methods have recently been considered and applied to the Set Covering Problem (SCP) with promising results, for instance see [4, 15, 20, 27].

Since it is computationally prohibitive to find the optimal solution among the feasible set of solutions, we use genetic algorithms (GA) for solving the second phase in order to find an

approximate solution for large size problems. GA, which mimics the natural biology evolution, uses various selection criteria to generate a ‘good’ solution from a population of neighboring solutions. The population of neighboring solutions is updated in each iteration in order to force the algorithm out of the local solution. We next describe the use of GA to generate an approximate solution for the second phase.

III.2.1. Methodology:

First, the sets of the routes found in Phase I are encoded in the form of matrix β presented in (12). Matrix β 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 β_{ij} of this matrix has a binary value, i.e., it is either one or zero. The β_{ij} is one if route j visits node $i \in N$, and zero otherwise. Without loss of generality, we assume that the columns of β 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 (13) illustrates a typical matrix β for a graph of 4 nodes where 11 feasible routes were generated in Phase I.

$$\beta = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

A solution to the set covering problem is a set of routes that visit all the customers exactly once. Using the encoded representation of feasible routes in (13), a solution will be a set of columns in matrix β . For example a solution to matrix β in (13) would be the set consisting of columns 2 and 8; another solution would be the set of columns 3 and 4. Note 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 hereafter, 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. For instance, for the example given above, the chromosome 1, *chrml* in Equation (14), represents a solution to matrix β in (13) which consists of the set of columns 2 and 8; likewise chromosome 2, *chrml2*, presents a solution consisting of columns 3 and 4 in matrix β .

$$\begin{aligned} \text{chr}m1 &= [0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0] \\ \text{chr}m2 &= [0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \end{aligned} \quad (14)$$

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

$$\psi_j = \sum_{i=1}^m F_i \cdot \rho_{ij} \quad (15)$$

where ρ_{ij} is the value of the i th bit in chromosome j , and F_i is the cost of traveling on route i as 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. The generated chromosomes may not be feasible. That is the summation over the rows of corresponding columns in generated chromosomes may not always be a unit vector column. Thus, a heuristic method is developed to make the initial chromosomes feasible. This method is discussed later in this section.

For our problem of interest, the m-TSPTW problem with social constraints, the number of chromosomes in the population is selected such that there would be enough good neighboring solutions in the population. More precisely, we select the initial population to be 8 when the number of customers is 10 or less, 12 when it is between 11 to 20, 20 when it is between 21 to 50, and 30 when there is more than 51 customers.

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 almost equal number of chromosomes in each group. The chromosomes are randomly placed in each group. The chromosome from each group with the minimum fitness function value 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 ρ_{il} (the i th bit in chromosome l) is set equal to ρ_{ij} with probability p , which is equal to

$$p = \frac{\psi_j}{\psi_j + \psi_k}, \quad (16)$$

and to ρ_{ik} with probability $(1-p)$. Obviously, if the values of ρ_{ij} and ρ_{ik} are equal, the value of ρ_{il} will be equal to this value, i.e., $\rho_{il}=\rho_{ij}=\rho_{ik}$. By applying the mutation operator, the value of ρ_{il} will be inverted (zero to one, and one to zero) by some small probability q .

The new generated offspring (i.e., the l th chromosome) may not be feasible. As mentioned earlier, for a solution to be feasible the summation over the rows of the corresponding columns should generate a unit vector column. The crossover and mutation operators applied to the offspring, chromosome l , may not have preserved this characteristic. Thus, a heuristic method is developed to make the offspring feasible. The method not only maintains the feasibility but also provides a local minimization.

In this method, routes (columns) with higher number of covered customers (ones in their corresponding rows) will be given higher priority to stay in the offspring. More precisely, the heuristic method starts by searching for the smallest bit i in the chromosome l whose corresponding value is 1, i.e. the first i with $\rho_{il} = 1$. The column i in chromosome l covers a number of customers (has value 1 in some rows). The corresponding rows with value 1 are obtained. The algorithm *keeps* column (bit) i in the chromosome l (i.e., the value of ρ_{il} will not be changed) and searches for any other bit j in chromosome l whose value is 1 (i.e., $\rho_{jl} = 1$) and its corresponding rows overlap with rows of bit i . If such a bit is found, its value, ρ_{jl} , will be set to zero. The algorithm then searches for the next smallest bit i in the chromosome l with $\rho_{il} = 1$, and the same procedure continues. The outcome of the algorithm will be a set of columns in chromosome l covering “some” of the customers (rows) exactly once. Note that some of the customers (rows) will not be served by a set of columns in l . We call these customers *remaining* customers.

Columns, which cover the remaining rows only, are retrieved from matrix β . Among these retrieved columns, one is randomly selected to be added to the l th chromosome. The retrieved column i is weighted more if it covers more of the remaining nodes with less total costs. The procedure continues till all customers are covered exactly once by columns kept or added to chromosome l .

The newly generated offspring replaces one of the chromosomes in the initial population with a fitness function value 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 β ; generate the initial population; find the chromosome with the minimum fitness function value. This is the *best solution so far*.

Step 2: Select two chromosomes for generating an 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. If the fitness function value is better than the value of the *best solution so far*, substitute the newly found solution (chromosome) with the *best solution so far*.

Step 7: Replace randomly one of the chromosomes in the initial population with the offspring.

Step 8: Repeat steps 2-8 until the termination criterion is met. The best solution is the *best solution so far*.

It is generally believed that GA is slow and would take time to find a high quality solution [33]. 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 β) and the number of the generated feasible solutions in Phase I (number of the columns in matrix β).

III.2.2. 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.

| No of nodes | Best solution in each of 10 trials using GA | | | | | | | | | | Best Value/ CPU Time | |
|-------------|---|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-------------------------|-----------------|
| | | | | | | | | | | | Mean | SD ^b |
| 7 | 13.69 ^a / 0.39 | 13.69/ 0.16 | 13.69/ 0.22 | 13.69/ 0.16 | 13.69/ 0.17 | 13.69/ 0.11 | 13.69/ 0.16 | 13.69/ 0.22 | 13.69/ 0.28 | 13.69/ 0.38 | 13.69/ 0.225 | 0/ 0.095 |
| 10 | 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 | 23.88/ 0.501 | 0/ 0.5 |
| 15 | 35.38/ 0.88 | 35.38/ 0.77 | 35.38/ 0.77 | 35.38/ 0.33 | 35.38/ 0.72 | 35.38/ 1.16 | 35.38/ 1.7 | 35.38/ 0.5 | 35.38/ 0.33 | 35.38/ 0.5 | 35.38/ 0.766 | 0/ 0.42 |
| 20 | 52.38/ 5.76 | 52.39/ 33.45 | 52.38/ 15.21 | 52.39/ 28.28 | 52.93/ 2.86 | 52.38/ 9.5 | 52.38/ 13.34 | 52.38/ 12.86 | 52.38/ 10.39 | 52.39/ 26.96 | 52.44/ 15.86 | 0.02/ 10.2 |
| 30 | 100.3/ 31.97 | 98.21/ 26.86 | 100.3/ 45.65 | 97.19/ 131.8 | 99.3/ 55.31 | 100.3/ 39.7 | 99.3/ 124.5 | 98.21/ 35.75 | 99.3/ 95.1 | 97.19/ 29.22 | 98.97/ 61.58 | 1.46/ 40.25 |
| 50 | 179.0/ 85.62 | 179.1/ 234.6 | 174.2/ 89.7 | 180.2/ 239.2 | 186.7/ 204.4 | 173.9/ 157.1 | 175.8/ 255.9 | 180.9/ 203.5 | 181.2/ 178.4 | 182.3/ 270.6 | 179.4/ 191.9 | 15.36/ 64.8 |
| 100 | 337.8/ 1865 | 337.0/ 2062 | 339.7/ 2179 | 340.6/ 1876 | 337.1/ 2079 | 340.7/ 2232 | 336.6/ 1678 | 342.1/ 1806 | 338.0/ 1415 | 339.8/ 1569 | 338.9/ 1876 | 3.55/ 266.4 |

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

b) SD: Standard Deviation.

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 the 10 trials.

Table 3: The hybrid GA method: the summary of the computational experiments for different window size (based on the results of 10 trials).

| No of nodes | w=0.5 hour | | w=1 hour | | w=2 hours | | w=3 hours | |
|-------------|------------|----------|-----------|----------|-----------|----------|-----------|----------|
| | Best Cost | CPU time | Best Cost | CPU time | Best Cost | CPU time | Best Cost | CPU time |
| 7 | 15.10 ° | 0.20 | 15.43 ° | 0.18 | 13.69 ° | 0.22 | 13.69 ° | 0.27 |
| 10 | 28.82 ° | 0.19 | 28.82 ° | 0.19 | 23.88 ° | 0.5 | 23.16 ° | 0.9 |
| 15 | 48.49 ° | 0.59 | 48.49 ° | 0.99 | 35.38 ° | 0.76 | 37.70 ° | 6.34 |
| 20 | 66.00 ° | 2.17 | 66.00 ° | 2.89 | 52.44 | 15.86 | 48.97 | 11.71 |
| 30 | 112.2 | 28.76 | 104.0 | 23.64 | 98.97 | 61.61 | 95.96 | 50.56 |
| 50 | 196.9 | 126.7 | 189.5 | 240.9 | 179.4 | 191.91 | 172.0 | 236.76 |
| 100 | 359.7 | 1234 | 352.4 | 1368 | 338.9 | 1876 | 329.5 | 1743 |

o) Optimum Value.

III.3. Insertion heuristic method for m-TSPTW with additional social constraints

Finally, we benchmark the proposed exact and hybrid genetic algorithm methods against a developed heuristic insertion method similar to that presented in [16]. Solomon in [29] described a variety of heuristics for the VRPTW. He conducted an extensive computational study of evaluating the performance of each heuristic. He found that several heuristics performed well in different problem environments; however, in particular an insertion-type heuristic consistently gave very good results. We next briefly describe the developed insertion heuristic method.

III.3.1. Methodology:

Let $r = \{o, 1, \dots, i, j, j+1, \dots, n, d\}$ be a feasible solution (route) starting from origin $\{o\}$, visiting nodes in route r only once, and ending at destination $\{d\}$. Let A^r be the arc set associated with route (solution) r defined as $A^r = \{(i, j) / i, j \in r, \text{ and node } j \text{ is visited immediately after node } i\}$. Let also t_i be the time when service begins at node i . Recall that associated to each node $i \in r$ are a service time s_i and a time window $[a_i, b_i]$, and to each arc $(i, j) \in A^r$ are a cost c_{ij} and a time t_{ij} representing the cost and the time of traveling between nodes i and j , respectively.

Assuming that node j is visited after node i , the *waiting time* at node j denoted by w_j is defined as the duration of time at node j a vehicle has to wait before being served at that node, and is given by

$$w_j = \max(0, a_j - (t_i + s_i + t_{ij})) \quad (17)$$

Similarly, the *excess time* at node i denoted by Δ_i is defined as the portion of the time window (at node i) between the latest time to visit node i , b_i , and the time that the service has started at node i , and is given by

$$\Delta_i = b_i - \max(t_i, a_i) \quad (18)$$

Figure 5 illustrates the waiting and excess times at nodes of route r graphically. We define the *feasibility margin* of the solution r with respect to the changes in the traveling time on the link (i,j) , denoted by ϕ_{ij}^r , as the maximum value of the disturbance in the traveling time on that link such that the solution r will still be feasible. ϕ_{ij}^r can be computed as follows:

$$\phi_{ij}^r = \underset{k \in \{j, j+1, \dots, n, d\}}{\text{Min}} \left(\Delta_k + \sum_{l=j}^k w_l \right) \quad (19)$$

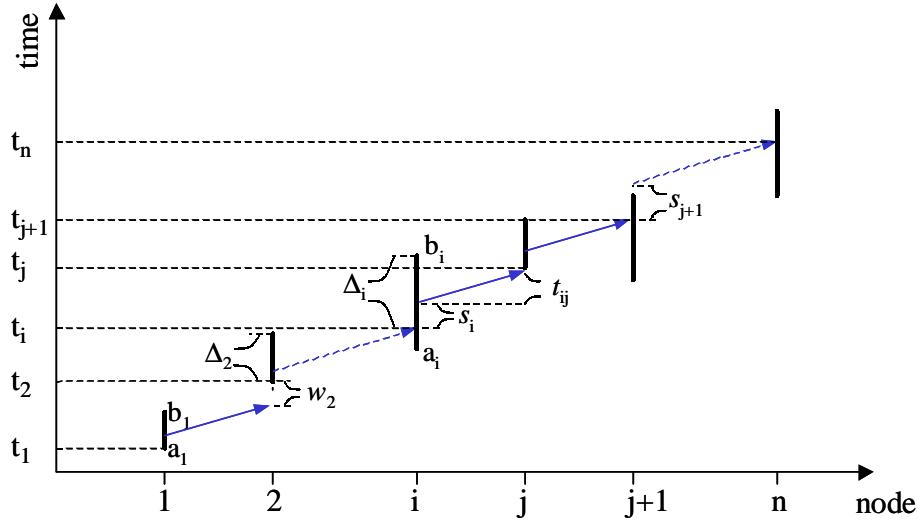


Figure 5: The graphical representation of waiting time, excess time, service time, and time windows at each node in a typical route.

The feasibility margin ϕ_{ij}^r implies how robust is the solution r to the changes on the traveling time between nodes i and j . The feasibility margin in (19) can also be calculated recursively as follows:

$$\begin{aligned}\varphi_{ij}^r &= w_j + \text{Min}(\Delta_j, \varphi_{j,j+1}^r) \\ \varphi_{d,d+1}^r &= M\end{aligned}\tag{20}$$

where $j+1$ is the next node to be visited on route r after node j , and M is a big number. Node l can be inserted between each two consecutive nodes i and j on route r if the following inequalities are satisfied,

$$t_{il} + w_l + s_l + t_{lj} - t_{ij} \leq \varphi_{ij}^r\tag{21}$$

$$t_i + s_i + t_{il} \leq b_l\tag{22}$$

and the total traveling time on route r (including node l) is less than a certain amount of time, working shift T . Equation (21) indicates that the changes in time on route r caused by inserting node l between nodes i and j should be less than or equal to φ_{ij}^r . Equation (22) ensures that node l is visited before the latest time b_l .

Insertion procedure: To each route r a cost F^r is associated which is the summation of the traveling costs between each two consecutive nodes on route r . In other words, F^r is the total cost of visiting all nodes of route r in the specified order. Let's assume that node l can be inserted between nodes i and j on route r (i.e., inequalities (21) and (22) are met). The changes in the cost F^r due to the insertion of node l between nodes i and j is denoted by ΔF_{ij}^r and is given by

$$\Delta F_{ij}^r = c_{il} + c_{lj} - c_{ij}\tag{23}$$

To insert a node l in route r , we first find all feasible insertion locations according to (21) and (22). Among all feasible locations (arcs) on route r , the *candidate* arc $(i,j) \in A^r$ is the one which leads to minimum changes in cost F^r . In other words, the *candidate* is the arc obtained by

$$\text{Min}_{(i,j) \in A^r} \Delta F_{ij}^r\tag{24}$$

We then examine all routes r to find the best location among all feasible locations to insert node l . That is, we are interested in finding the best candidate among all candidates with minimum cost of insertion:

$$\text{Min}_r \left\{ \text{Min}_{(i,j) \in A^r} \Delta F_{ij}^r \right\} \quad (25)$$

If such a candidate couldn't be found, a new route would be initialized. The node l will then be added to the newly generated route. The algorithm will be terminated when no more nodes are left to be inserted.

III.3.2. Computational Experiments:

Simulation scenario 1: The same sets of generated customers in Table 3, for different number of customers in graph G , are used for evaluating the insertion method. 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. The time window length is 2 hours. As shown in Table 4, the exact method is efficient for relatively small size problems consisting of a few nodes. GA is also capable of finding the optimum solution for small size problems, and significantly outperforms the insertion heuristic method when the number of nodes is less than 100.

Table 4: Comparing exact, hybrid GA, and insertion methods, $w=2$ hours.

| No of nodes | Dynamic Programming | | Genetic Algorithm ^a | | Insertion method | |
|-------------|---------------------|----------|--------------------------------|----------|------------------|----------|
| | Cost | CPU time | Cost | CPU time | Cost | CPU time |
| 7 | 13.69 | 0.38 | 13.69 | 0.22 | 17.69 | 0.11 |
| 10 | 23.88 | 1.70 | 23.88 | 0.50 | 26.96 | 0.11 |
| 15 | 35.38 | 326.4 | 35.38 | 0.76 | 42.25 | 0.16 |
| 20 | NA ^b | NA | 52.44 | 15.86 | 59.91 | 0.27 |
| 30 | NA | NA | 98.97 | 61.61 | 107.0 | 0.49 |
| 50 | NA | NA | 179.4 | 191.91 | 194.45 | 1.45 |
| 100 | NA | NA | 338.9 | 1876 | 359.08 | 4.67 |

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) for the GA was limited to 1000. Obviously, by increasing this number a better solution may be found.

Simulation scenario 2: (Real life simulation scenario) The developed algorithms were tested using real examples containing a truck depot, container terminals, intermodal facilities and end customers supplied by a trucking company, Transport Express, in the City of Los Angeles. The city has some unique specifications. It is adjacent to the ports of Los Angeles and Long Beach (LA/LB), and the Intermodal Container Transfer Facility (ICTF). Transport Express is a trucking company, which is located 6 to 10 miles north of the LA/LB container terminals.

The real data were obtained through observing the dispatcher within a typical day in 2001. It was observed that the dispatcher assigned the jobs to drivers based on the availability of the drivers and jobs. The priority in job assignments went to those drivers who showed up early at the depot. The dispatcher made sure that the jobs are distributed evenly without concerns of optimizing a cost function. During the day, 65 containers were assigned to 22 drivers. This number includes 42 containers, which were moved between the depot and container terminals. This number showed the amount of similar activities at the depot.

Time windows for container delivery were very large, i.e., between 7:00 a.m. to 4:00 p.m. Only 10 jobs were associated with tighter time windows either before noon or after noon. This fact of having 65 nodes with wide time windows would actually render both the exact and hybrid GA methods incapable of demonstrating any results. Thus, for this data set only the insertion method could find a solution. The plan to move to smaller time windows would make the exact and GA methods viable solution techniques as illustrated by the earlier experiments.

The data obtained from the company consists of the containers' pick up and drop off locations, the assigned drivers, and the time windows at origins and/or destinations. After finishing his current job, a driver might be given his next job via a cell phone. We noticed that at the time of the next job assignment the driver might have been at the current drop-off location, on his/her way back to the depot, or even in the yard of the company. In other words, it was hard for us to locate the exact location of the drivers when he/she received his/her next assignment via a cell phone. Therefore, for the purpose of the simulation, we build two cases: the worst-case and best-case scenarios. In the worst-case scenario, we assume that all drivers were back at the depot at the time of the next job assignment. In the best-case scenario, we assume that the drivers were given the next job assignment as soon as they dropped off their loads at the customers' locations.

Table 5 shows the results obtained by applying the insertion method to the set of real data obtained from Transport Express. The cost is set equal to the traveling distance. Note that the traveling distance is chosen for cost comparison since the available data from Transport Express doesn't carry the exact traveling time for each trip, the service time at pick-up and drop-off locations, etc.

We assume that the loading and unloading times at the customers' and/or STs' locations is 20 minutes. In addition, we assume that the traveling speed of the trucks is 30 miles per hour.

Table 5. Real-data solutions comparison.

| No. of nodes | Insertion method | | | Dispatcher Decision | | |
|--------------|------------------|--------------|----------|-------------------------|------------------------|--------------|
| | Cost [miles] | No. Vehicles | CPU Time | Cost [miles] Worst-case | Cost [miles] Best-case | No. Vehicles |
| 65 | 735 | 14 | 1.20 | 1057 | 825 | 22 |

Table 5 indicates that the routes generated by the insertion method outperform those decided by the dispatcher in two dimensions: cost and the number of vehicles used. It also shows that for the current scenario (very wide time windows), the insertion method is capable of generating good solutions in a fairly short amount of time.

IV. CONCLUSIONS

In this paper, we investigated the cargo movement problem in metropolitan areas adjacent to marine ports. In particular, we were interested in improving the method 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 problem by trucks with time constraints at both origins and destinations could be modeled as an asymmetric multi-Traveling Salesmen Problem with Time Windows (m-TSPTW) with additional social constraints.

Furthermore, we developed and compared the following three methodologies for solving the m-TSPTW with additional social constraints:

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

The results of our computational experiments indicate that the exact method was efficient for relatively small size problems (compared to the real-life problem discussed earlier) 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). The insertion heuristic method was able to find relatively good solutions for large size problems; and the method was computationally very fast.

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

- [1] N. Ascheuer, M. Fischetti, and M. Grottschel, "A Polyhedral Study of the Asymmetric Traveling Salesman Problem with Time Windows," *Networks*, vol. 36, no. 2, pp. 69-79, 2000.
- [2] D. Barber, and L. Grobar, "Implementing A Statewide Goods Movement Strategy and Performance Measurement of Goods Movement in California," Technical Report, Metrans Report 99-10, June 2001.
- [3] M. E. Barton, "24/7 Operation by Marine Terminals in Southern California: How to Make it Happen," CITT Industry Stakeholder Workshop One, Metrans Report, Nov. 2001.
- [4] J.E. Beasley, and P.C. Chu, "A Genetic Algorithm for the Set Covering Problem," *European Journal of Operational Research*, vol. 94, pp. 392-404, 1996.
- [5] L. Bodin, B. Golden, A. Assad, and M. Ball, "Routing and scheduling of vehicles and crews: the state of the art," *Computers & Operations Research*, vol. 10, no. 2, pp. 63-211, 1983.
- [6] R.W. Cavo, "A New Heuristic for the Traveling Salesman Problem with Time Windows," *Transportation Science*, vol. 34, no. 1, pp. 113-124, 2000.
- [7] M. Desrochers, J. Desrosiers, and M. Solomon, "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows," *Operations Research*, vol. 40, pp. 342-354, 1992.

- [8] J. Desrosiers, Y. Dumas, M.M Solomon, and F. Soumis, "Time Constrained Routing and Scheduling", in: M.O. Ball, T.L. Magnati, C.L. Monma, and G.L. Nemhauser, (eds.) *Network Routing Handbooks in Operations Research and management Science*, Volume 8, INFORMS, Elsevier Science, pp. 35-130, 1995.
- [9] Y. Dumas, J. Desrosiers, E. Gelinas, M.M. Solomon, "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows," *Operations Research*, vol. 43, no. 2, pp. 367-371, 1995.
- [10] M. Fisher, "Vehicle Routing," in: M.O. Ball, T.L. Magnati, C.L. Monma, and G.L. Nemhauser, (eds.) *Network Routing Handbooks in Operations Research and management Science*, vol. 8, INFORMS, Elsevier Science, pp. 1-33, 1995.
- [11] F. Focacci, A. Lodi, and M. Milano, "A Hybrid Exact Algorithm for the TSPTW," *INFORMS Journal on Computing* vol. 14, pp. 403-417, 2002.
- [12] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.
- [13] M. Gendreau, A. Hertz, G. Laporte, and M. Stan, "A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows," *Operations Research*, vol. 43, no. 3, pp. 330-335, 1998.
- [14] B.L. Golden, and A.A. Assad, *Vehicle Routing Methods and Studies*, North Holland Publication, Amsterdam, 1988.
- [15] L.W. Jacobs, and M.J. Brusco, "A Simulated Annealing Based Heuristic for the Set-Covering Problem," *Proceedings of Decision Sciences Institute*, vol. 2, pp. 1189-91, 1994.
- [16] J.J. Jaw, A.R Odoni., H.N. Psaraftis, and N.H.M. Wilson, "A Heuristic Algorithm for the Multi-Vehicle Advance Request Dial-A-Ride Problem with Time Windows," *Transportation Research - Part B*, vol. 20, no. 3, pp. 243-257, 1986.
- [17] N. Kohl, J. Desrosiers, O.B.G. Madsen, M.M. Solomon, and F. Soumis, "2-Path Cuts for the Vehicle Routing Problem with Time Windows," *Transportation Science*, vol. 33, no. 1, pp. 101-16, 1999.
- [18] M.S. Lee, New Algorithms for the m-TSPTW, Ph.D. Thesis, University of Maryland College Park, 1992.
- [19] S. Lin, and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 1, pp. 498-516, 1973.
- [20] L. Lorena, and L.S. Lopes, "Genetic Algorithms Applied to Computationally Difficult Problems," *Journal of Operational Research Society*, vol. 48, pp. 440-445, 1997.
- [21] L.G. Mallon, and J.P. Magaddino, "An Integrated Approach to Managing Local Container Traffic Growth in the Long Beach –Los Angeles Port Complex, Phase II", Technical Report, Metrans Report 00-17, Dec. 2001.

- [22] Meyer, Mohaddes Associates, Inc., "Gateway Cities Trucking Study," Gateway Cities Council of Governments Southeast Los Angeles County, 1996.
- [23] L. Ng, R.L. Wessels, D. Do, F. Mannering, and W. Barfield, "Statistical Analysis of Commercial Driver and Dispatcher Requirements for Advanced Traveller Information Systems," *Transportation Research – Part C*, vol. 3, no. 6, 353-369, 1995.
- [24] G. Reinelt, *The traveling salesman problem: computational solutions for TSP applications*, Lecture Notes in Computer Science, Springer-Verlag, 1994.
- [25] M.W.P. Savelsbergh, "Local Search in Routing Problems with Time Windows," *Annals of Operations Research*, vol. 4, no. 1-4, pp. 285-305, 1985.
- [26] M.W.P Savelsbergh, and N. Sol, "The General Pickup and Delivery Problem," *Transportation Science*, vol. 29, no. 1, pp. 17-29, 1995.
- [27] S. Sen, "Minimal Cost Set Covering Using Probabilistic Methods," *Proceeding of 1993 ACM/SIGAPP Symposium of Applied Computing* pp. 157-164, 1993.
- [28] P. Senecal, (Private Communication), Transport Express Inc., Rancho Dominguez, CA, 2001.
- [29] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research*, vol. 35, no. 2, pp. 254-265, 1987.
- [30] The Tioga Group, "Empty Ocean Logistics Study," Technical Report, Submitted to the Gateway Cities Council of Governments, May 2002.
- [31] M.J. Vickerman, "Next-Generation Container Vessels: Impact on transportation Infrastructure an Operations," *TR News*, vol. 196, pp. 3-15, May-June 1998.
- [32] X. Wang, and A.C. Regan, "Local Truckload Pickup and Delivery with Hard time Window Constraints," *Transportation Research – Part B*, vol. 36, pp. 97-112, 2002.
- [33] A.M.S. Zalzal, and P.J. Fleming, *Genetic Algorithms in Engineering Systems*, IEE control Engineering Series 55, 1997.