Working Paper

The Time-Dependent Multiple-Vehicle Prize-Collecting Arc Routing Problem

Dan Black*

University of Edinburgh Business School, 29 Buccleuch Place, Edinburgh, EH8 9JS, U.K.

Richard Eglese

Department of Management Science, Lancaster University Management School, Lancaster, LA1 4YX, U.K.

Sanne Wøhlk[1]

CORAL - Cluster for OR and Logistics, Department of Economics and Business, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark

Abstract

In this paper, we introduce a multi vehicle version of the Time-Dependent Prize-Collecting Arc Routing Problem (TD-MPARP). It is inspired by a situation where a transport manager has to choose between a number of full truck load pick-ups and deliveries to be performed by a fleet of vehicles. Real-life traffic situations where the travel times change with the time of day are taken into account.

Two metaheuristic algorithms, one based on Variable Neighborhood Search and one based on Tabu Search, are proposed and tested for a set of benchmark problems, generated from real road networks and travel time information. Both algorithms are capable of finding good solutions, though the Tabu Search approach generally shows better performance for large instances whereas the VNS is superior for small instances. We discuss the structural differences of the implementation of the algorithms which explain these results.

*Corresponding author
[1]Supported by NordForsk Project No. 25900.

1. Introduction

The problem under consideration is motivated by freight transport companies with potential orders for full truck loads to be carried between pairs of pick up points and destinations. A fleet of homogeneous vehicles is available at a depot to carry out the jobs. All potential orders are known before any of the vehicles leave the depot and the potential orders may be accepted or rejected by the transport company. When the company fulfils an order by carrying out a job, then the company receives a benefit or prize of a known value.

The objective is to maximize the sum of the prizes from the accepted orders minus a travel cost proportional to the time taken to complete the accepted orders and return to the depot.

We model this problem as a directed arc routing problem with prizes where each job corresponds to a profitable arc and the prize is collected the first time the arc is traversed. No further reward is obtained by a subsequent traversal of a prize arc.

Travel times between locations may change according to the time of day and day of week. This information is stored in an array known as a Road Timetable [12] and the vehicles must complete their jobs and return to the depot within a fixed time limit.

We define the Time-Dependent Multiple Vehicle Prize-Collecting Arc Routing Problem (TD-MPARP) as follows.

Problem 1 (TD-MPARP). *A directed multigraph $G(V, A)$ with a particular node designated as the depot node is given. Let the depot contain a set Q of identical vehicles, let $\mathcal{P} \subseteq A$ be a set of n profitable arcs and let $p_r \geq 0$ be a prize associated with arc $r \in \mathcal{P}$. This prize is collected the first time the arc is traversed. Furthermore, let $f_t(i, j)$ be the time it takes to travel from node i to node j starting at time t and let $f_t(r)$ be the time it takes to traverse the r'th profitable arc starting at time t. The goal is to construct a tour for each vehicle starting from the depot at time $T_{\min}$ and ending at the depot node by time $T_{\max}$, such that the sum of prizes collected minus the total travel cost is maximized. Waiting is not permitted initially at the depot node, nor at any node on the route.*

In [7] we considered a single vehicle version of this problem and presented a VNS algorithm and a Tabu Search algorithm for solving it. In this paper, we show how these algorithms can be extended to handle the multi–vehicle problem as described above. The performances of the algorithms will be tested on sets of benchmark problems and the results will be compared. The aim of the experiments is to discover whether there are any significant differences in the performances of the two algorithms for problems of different size and when different amounts of computing time are available. The rest of the paper is outlined as follows. Section 2 contains a review of related literature and

Section 3 provides a formal mathematical formulation of the problem. Section 4 describes the solution procedures and Section 5 describes the results from the computational experiments. The final section contains some concluding remarks.

2. Related Literature

Incorporating prizes or profitability within logistics problems has received an increasing amount of attention. These problems are seen to reflect the real-world situation where the amount of work available is beyond the capacity of the vehicle fleet and some form of prioritization must take place. They move beyond standard models, whether vehicle or arc routing, by incorporating two objectives: prizes and travel cost. Within the Prize-collecting Arc Routing Literature (PARP), these two objectives are usually combined into a single measure of profit which is then maximized. A small number of papers take the approach of maximizing only prizes subject to a cost constraint. No papers, to our knowledge, minimize cost subject to a minimum prize constraint although an equivalent problem has been examined within the prize-collecting Vehicle Routing Problem (VRP) literature [10]. This section will consider several key features of the PARP and how they are addressed within the literature.

2.1. Types of Prize

Within the Prize-collecting Arc Routing Problem literature, the nature of the prizes collected takes several forms. In this subsection we shall provide examples of prize types from the literature and then provide a broad categorization of problems into *efficiency oriented* and *overloaded* problems.

The Privatized Rural Postman Problem (PRPP) is introduced by Aráoz et al. [3]. The prizes in this problem consist of a single prize that cannot be collected on repeated traversals of the associated arc. The paper presents a polyhedral model and examines problem cases based on specific graph structures. This problem is subsequently solved by Aráoz et al. [2] using a 2-phase algorithm. At each phase, relaxations are solved using a heuristic and cuts are added to the problem. If the LP relaxation used in the first phase does not find the optimal solution, the algorithm resorts to the IP formulation of the second phase. An equivalent problem, called the Profitable Arc Tour Problem, that allows the prize associated with an edge to be collected more than once is presented in Feillet et al. [14]. This problem is based on a logistics problem where stock is moved between sites on a daily basis. The prize associated with each edge corresponds to the saving of not using a "there and back" trip (which is how the stock must be moved if there are uncollected prize arcs). The nature of this problem means that there is no depot. Euchi et al. apply a meta-heuristic to the same problem but with no correlation between arc length and prize value

[13]. A clustered variation of the PRPP is introduced by Aráoz et al. [1]. In this case there is again a single prize per arc, but arcs are clustered geographically and the prize of an arc can only be collected if all arcs of the corresponding cluster are traversed. The clustering requirement leads to new cuts that can be incorporated into the branch and bound search.

The Maximum Benefit Chinese Postman Problem (MBCPP) is introduced by Malandraki and Daskin [21]. In this variation of the PARP, a prize can be collected from an arc on more than one traversal. The first traversal will collect an initial prize, the next traversal a lower prize, the next an even lower prize, and so on. The multiple prizes of this problem correspond to real world problems such as ploughing snowy streets, where repeated traversals of a road will have some extra benefit. The problem is formulated as a minimum cost flow problem and branch and bound is combined with sub-tour elimination cuts to find an optimal solution. The MBCPP as defined in [21] is a directed problem. The undirected variant is examined by Pearn and Wang [24] who propose a heuristic solution method.

The nature of these prizes affects the overall problem objective such that Prize-collecting Arc Routing Problems can be broadly categorized into *efficiency oriented* and *overloaded problems*. An example of an *efficiency oriented* problem is the MBCPP of Malandraki and Daskin [21]. There is no bound on the total amount of work or effort that can be spent when collecting the prizes. The problem is concerned with determining at what point repeated traversal of prize arcs is no longer profitable. A similar situation occurs for The Profitable Arc Tour Problem of Feillet et al. [14]. In this case, there is a limit on the length of prize collecting tours; however, there is no limit on the number of tours. Prizes of arcs will remain uncollected if there is no profit in traversing them. The clustered problem introduced by Aráoz et al. will only collect the clusters of prizes if the overall profit is positive [1].

On the other hand, *overloaded* problems have too many arcs with profitable prizes. This is due to a constraint on total effort. For example, the PARP as defined in Black et al. has a constraint on driver shift time that limits the number of arc traversals within a route [7]. If this limit were not in place, then more arcs would be traversed and more prizes could be collected. A similar problem is considered by Archetti et al. [4]. In this case, there are two bounds on effort: travel time and total capacity related to the arcs where prizes are collected. The objective of [4] is to maximize the total of the prizes collected. This is in contrast to the usual maximization of profit. This work is extended in [5] to consider both prize collecting arcs and arcs that must be traversed. This problem is then solved using branch and cut. Zachariadis and Kiranoudis extend the problem studied in [4] to include a secondary objective of minimizing travel time [30]. Malandraki and Daskin examine the overlap of these definitions by varying a time constraint added to an *efficiency oriented* problem and produce a Pareto curve to show the relationship between the two criteria [21].

2.2. Variable Travel Times

One way that variable travel times or costs have been incorporated into Arc Routing Problems is through the so-called *windy* graphs. In this case, the travel time along an arc is dependent on the direction in which the arc is being traversed. This type of problem structure was originally introduced to non-prize collecting Arc Routing Problems; however, Corberán et al. [8] apply the *windy* extension to the Clustered PARP of Aráoz et al. [1]. They produce a number of inequalities for the problem that can be used as cuts within a branch and bound solution approach.

A more recent development within the routing literature has been the use of time-dependent travel times. These allow traffic congestion to be included within models and have become possible due to increased access to real-world traffic data. Ichoua et al. incorporate time-dependent travel times into the Vehicle Routing and Scheduling Problem [18]. They introduce the First In First Out or FIFO property of these travel times. This states that given a variable journey time from location A to location B, then if vehicle 1 leaves A after vehicle 2, then vehicle 1 cannot overtake vehicle 2. Eglese et al. use this property when constructing a *Road Timetable* of time-dependent journey times between the locations of a Capacitated Vehicle Routing Problem [12]. This is extended in Maden et al. who use a case study and real-world traffic data [20]. This study determines the benefit that can be obtained from using traffic data and also includes a comparison of the CO2 emissions made by vehicles. Both of these papers use meta-heuristics to find solutions to the problem. Calculating the cost of neighborhood moves can be a laborious process when using time-dependent travel data. The work of Harwood et al. examines the potential of using estimates for these calculations [17]. An exact approach to solving time-dependent VRPs was initially proposed by Soler et al. [28]. This involves transforming the the problem into an Asynchronous Capacitated VRP, however, no analysis of the performance of the method is provided. More recently, Dabia et al. have presented a branch and price approach to solving a variation of the problem where the starting time for each vehicle is a variable [9].

2.3. Full Truckload Transportation

A group of problems related to the PARP are the Full Truckload Transportation Problems (FTTP's). These problems consist of a Pickup and Delivery Problem (PDP) where a set of items must be picked up from certain locations and delivered to other customer locations, yet only one item can be carried by a vehicle at any one time. The obvious difference between these problems and PARP is that for the FTTP all deliveries must made. A number of other differences occur depending on problem definition. For example, the FTTP proposed by Gronalt et al. is motivated by the distribution of goods between distribution centers [15]. Such a problem will consist of moving multiple full truckloads rather than individual loads. This problem class distinction has also been highlighted by

Arunapuram et al. [6]. This type of problem may also be modeled with vehicles assigned to different depots. Most FTTP literature considers the problem as a variation of Vehicle Routing Problems; however, Liu et al. describe the problem as an Arc Routing Problem [19].

Several exact approaches to the FTTP have been proposed. Desrosier et al. redefine the problem as a Constrained TSP by collapsing Full Truck Load deliveries into a single node and modifying the associated journey times [11]. Arunapuram et al. use a Linear Programming relaxation that allows deliveries to be omitted to add cuts to a branch and bound search [6]. Heuristic approaches have also been suggested: Gronalt et al. extend the Clarke and Wright Savings Algorithm to the Full Truck Load problem [15]. Different two-phase heuristic approaches have been proposed by Liu et al. [19] and Nossack and Pesch [23] both consisting of a construction and a neighborhood improvement stage.

3. Mathematical Formulation

In this section, we present a mathematical model for the TD-MPARP. The model is an extension of the model for the single vehicle problem presented in Black et al. [7].

The problem is defined based on the directed multigraph $G(V, A)$. We consider a set of profitable arcs, $\mathcal{P}$ indexed by r. If there is potential to collect a prize from one of the original arcs multiple times, then the original arc is replaced by a set of similar arcs between the same pair of nodes, where the number of arcs between the pair of nodes corresponds to the maximum number of times the prize can be collected. For each such profitable arc $r \in \mathcal{P}$, we let $\alpha_r \in V$ be the source node, $\omega_r \in V$ the target node, and p_r the prize. For every node $j \in V$, we define $R_j^+ = \{r \in \mathcal{P} \,|\, \alpha_r = j\}$ to be the set of outgoing profitable arcs and $R_j^- = \{r \in \mathcal{P} \,|\, \omega_r = j\}$ to be the set of entering profitable arcs.

We consider a set Q of vehicles, each starting at the depot node d at time $T_{\min}$ and which must return to the depot by $T_{\max}$. For any pair of nodes, i and j, and any time t, we define the function $f_t(i, j)$ to be the time it takes to travel from node i to node j when starting at node i at time t and traveling along a shortest time route. For any profitable arc $r \in \mathcal{P}$, we denote by $f_t(r)$ the time it takes to traverse r starting at time t. In theory, f could be any continuous function, but in practice we will use the Road Timetable of Eglese et al. [12] to estimate this function.

In order to make a mathematical model, we restate the problem in terms of a node duplicated network $\tilde{G}(\tilde{V}, \tilde{A})$ as follows. For each node $j \in V$, we add $R_j^+ + R_j^-$ nodes to $\tilde{V}$. We refer to these nodes as copies of j and refer to j as the origin $h(k)$ of a copy k of j.

We first define the profitable arcs in $\tilde{G}$. For each profitable arc $r \in \mathcal{P}$, we select nodes k and l in $\tilde{V}$ such that k is a copy of α_r and l is a copy of ω_r. We add an

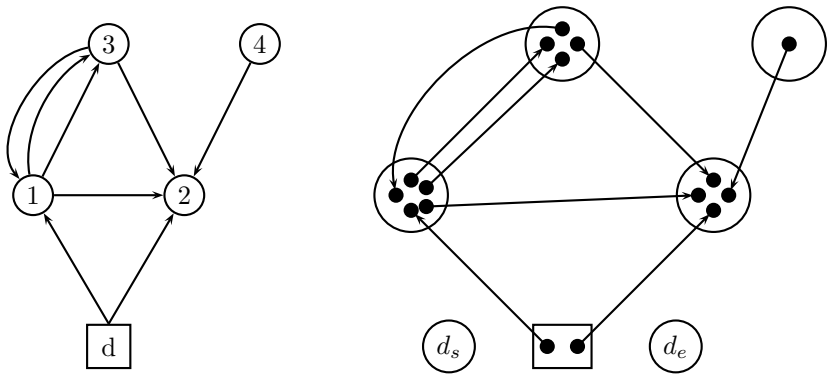

Figure 1: Example of instance (left) and the corresponding node duplicated network (right).

arc (k, l) to $\tilde{A}$ to be associated with the profitable arc. We select these nodes in $\tilde{V}$ in such a way that every node in $\tilde{V}$ is selected exactly once and therefore adjacent to exactly one profitable arc. We let $\tilde{\mathcal{P}}$ denote the set of these profitable arcs in $\tilde{G}$, and use $\tilde{\alpha_{\tilde{r}}}, \tilde{\omega_{\tilde{r}}}$, and $\tilde{p_{\tilde{r}}}$ to refer to the source node, target node, and profit of $\tilde{r} \in \tilde{\mathcal{P}}$, respectively. We define the travel time function for these arcs as $\tilde{f}_t(\tilde{\alpha_{\tilde{r}}}, \tilde{\omega_{\tilde{r}}}) = f_t(r)\ \forall t$.

Next, we define the deadheading arcs in $\tilde{G}$. For every pair of profitable arcs, $\tilde{r}$ and $\tilde{r'}$ in $\tilde{\mathcal{P}}$, we add deadheading arcs $(\tilde{\omega_{\tilde{r}}}, \tilde{\alpha_{\tilde{r'}}})$ and $(\tilde{\omega_{\tilde{r'}}}, \tilde{\alpha_{\tilde{r}}})$ to $\tilde{A}$. For each such deadheading arcs $(k, l) \in \tilde{A}$, we set the travel time function to $\tilde{f}_t(k, l) = f_t(h(k), h(l))$. It follows directly that $\tilde{f}_t(k, l) = 0$ if k and l are copies of the same node in G, i.e. if $h(k) = h(l)$.

Finally, we add two nodes, denoted by d_s and d_e to $\tilde{V}$. These nodes are copies of the depot and can be thought of as a super-source and a super-sink node. For every node $k \in \tilde{V} \setminus \{d_s, d_e\}$, we add deadheading arcs (d_s, k) and (k, d_e) with travel time functions $\tilde{f}_t(d_s, k) = f_t(d, h(k))$ and $\tilde{f}_t(k, d_e) = f_t(h(k), d)$, respectively. Note that the travel time function is zero, if k is a copy of the depot. Finally, we add an arc (d_s, d_e) with travel time function $\tilde{f}_t(d_s, d_e) = 0$.

The construction of $\tilde{G}(\tilde{V}, \tilde{A})$ is illustrated in Figure 1, where only profitable arcs are shown.

We use two types of decision variables for the construction of a mathematical model: For any arc (i, j) in $\tilde{G}$ and every vehicle, q, let x_{ij}^q be a binary variable taking the value of 1, if (i, j) is traversed by vehicle q and zero otherwise. Let t_{ij} be the time at which the traversal of the arc (i, j) starts. t_{ij} is defined for all arcs, but for arcs where x_{ij}^q is zero for all values of q, the value of t_{ij} is without interpretation. t_{ij} is independent of the vehicles.

$$\max \quad \sum_{q\in Q}\sum_{r\in\tilde{\mathcal{P}}} \tilde{p}_r x^q_{\tilde{\alpha}_r\tilde{\omega}_r} - \sum_{q\in Q}\sum_{(i,j)\in\tilde{A}} \tilde{f}_{t_{ij}}(i,j)x^q_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{q\in Q} x^q_{\tilde{\alpha}_r\tilde{\omega}_r} \leq 1 \quad \forall r \in \tilde{\mathcal{P}} \tag{2}$$

$$\sum_{j\in\tilde{V}\setminus\{d_s\}} x^q_{d_s j} = 1 \quad \forall q \in Q \tag{3}$$

$$\sum_{j\in\tilde{V}\setminus\{d_e\}} x^q_{j d_e} = 1 \quad \forall q \in Q \tag{4}$$

$$\sum_{j\in\tilde{V}} x^q_{ji} - \sum_{j\in\tilde{V}} x^q_{ij} = 0 \quad \forall i \in \tilde{V}\setminus\{d_s,d_e\}, \forall q \in Q \tag{5}$$

$$t_{id_e} + \tilde{f}_{t_{id_e}}(i,d_e)\sum_{q\in Q} x^q_{id_e} \leq T_{\max} \quad \forall (i,d_e) \in \tilde{A} \tag{6}$$

$$t_{ki} + \tilde{f}_{t_{ki}}(k,i)\sum_{q\in Q} x^q_{ki} \leq t_{ij} + (2 - \sum_{q\in Q}(x^q_{ki} + x^q_{ij}))M \quad \forall (k,i),(i,j) \in \tilde{A} \tag{7}$$

$$t_{ki} + \tilde{f}_{t_{ki}}(k,i)\sum_{q\in Q} x^q_{ki} \geq t_{ij} - (2 - \sum_{q\in Q}(x^q_{ki} + x^q_{ij}))M \quad \forall (k,i),(i,j) \in \tilde{A} \tag{8}$$

$$x^q_{d_s d_e} \geq 0, \text{integer} \quad \forall q \in Q \tag{9}$$

$$x^q_{ij} \in \{0,1\} \quad \forall (i,j) \in \tilde{A}\setminus\{(d_s,d_e)\}, \forall q \in Q \tag{10}$$

$$t_{ij} \geq T_{\min} \quad \forall (i,j) \in \tilde{A} \tag{11}$$

$$t_{d_s j} \leq T_{\min} \quad \forall (d_s,j) \in \tilde{A} \tag{12}$$

Here, (1) is the objective function maximizing the sum of the prizes minus travel time. Constraint (2) ensures that each prize is collected at most once. Constraints (3) and (4) ensure that all vehicles leave the super-source depot and enter the super-sink depot using exactly one arc, respectively. (5) is the flow conservation constraint and (6) ensures that all arrivals at the super-sink depot are on time. Constraint (7) ensures that we do not start traversing an arc before we have reached the source of the arc. M represents a sufficiently large number. This constraint is non-binding for any arc (i,j) that is not being traversed in the solution because the corresponding t_{ij} can take the value of zero. Constraint (8) ensures that the model does not allow for waiting in the nodes. Constraint (9) allows for vehicles to be left unused. Finally, (10) ensures binarity while (11) and (12) ensure that all start times are legal.

By the construction of the graph $\tilde{G}$, every feasible vehicle tour is alternating between an arc with a prize to be collected and a deadheading arc. As a consequence of this fact combined with constraint (2), the following inequality holds for any feasible solution:

$$\sum_{q \in Q} x_{ij}^q \leq 1 \quad \forall (i,j) \in \tilde{A} \setminus \{(d_s, d_e)\}.$$

The fact that any arc except possibly the one between the super-source and the super-sink is traversed at most once, explains why only one time variable, t_{ij} is needed for each arc.

We stress the fact that a key feature of the problem studied is that the time to travel from i to j is not constant, but rather it changes according to the time t at which the traversal takes place. This is modeled via the function $f_t(i,j)$ which changes with the parameter t. $f_t(i,j)$ directly influences the function $\tilde{f}_t(i,j)$, which appears both in the objective function and in constraints (6) through (8), resulting in a model which is not easily linearized.

4. Solution Procedure

We have developed two algorithms for solving the TD-MPARP: A Tabu Search algorithm called LANTIME and a Variable Neighborhood Search algorithm (VNS). Both algorithms are extensions of the algorithms constructed for handling the single vehicle problem presented in [7]. In the following, we will shortly outline the algorithms and focus on the parts of the algorithms which are different for this multi-vehicle problem.

4.1. LANTIME

LANTIME is a Tabu Search meta-heuristic that has been developed to tackle the time-dependent VRP (with no prizes). It has been modified to tackle TD-MPARP so as to provide a comparison for the VNS approach. The solution structure used, while differing from the representation used by VNS, is not uncommon. A potential solution is stored as a set of vehicle routes plus a dummy route. The implementation has been modified so that each route, rather than consisting of a set of deliveries to single customers, consists of a set of non-overlapping pick-ups and deliveries. The dummy route represents the prize arcs that will not be traversed by the solution.

The original implementation of LANTIME included 4 neighborhood moves. For the TD-MPARP empirical analysis has suggested that only the Insertion and Swap moves are required. These are chosen stochastically with the former selected 40% of the time and the latter 60%. LANTIME does not exhaustively search a given neighborhood but looks for a good move. A key modification is

required to ensure that there is a balance between moves that add jobs to the dummy route and that remove jobs from the dummy route. An infeasibility penalty is used to ensure diversity in search along with a Tabu list and memory structure to prevent dead-ends at local optima. More details of the Tabu search heuristic can be found in [20].

4.2. Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a meta-heuristic proposed by Mladenović and Hansen [22]. It has been used for a wide variety of optimization problems [16] and proved to be very good at solving the TD-PARP [7]. In modifying VNS for the TD-MPARP, the profitable arcs to be served will be ordered before splitting them into vehicle routes, i.e. a *route-first, cluster-second approach* is adopted.

In our VNS algorithm, we store all n profitable arcs in an ordered list L. For any $i, j \leq n, i < j$, we use $L[i:j]$ as short notation for the partial list $L[i], L[i+1], \ldots, L[j]$.

The intuition behind our algorithm is that at any given time, the current solution of the problem is determined by L by forcing the vehicles to service the jobs in the order dictated by L. This way, the first vehicle services jobs $L[1:i]$, the second vehicle services jobs $L[i+1:j]$, and so forth until the last vehicle services $L[k+1:s]$, where $i < j < k$ and s is a threshold. The value of s is determined such that $L[1:s]$ yields the solution with the highest objective value, among all values of s while giving the option of leaving some vehicles unused. That way, the profitable arcs $L[1:s]$ are accepted, while the profitable arcs in $L[s+1:n]$ are not serviced in the solution. We explain the algorithm for finding this optimal way of splitting $L[1:s]$ into vehicle routes in Section 4.2.1.

During the VNS algorithm, the order of L is changed, causing different solutions for the vehicles. As a subroutine of VNS we use a Variable Neighborhood Descent (VND) which uses a second threshold $s' = \min(s + \epsilon, n)$. During the experiments, $\epsilon = 5$ was found to be a good value. In the VND, we reorganize the jobs in $L[1:s']$, i.e. the jobs that are already accepted plus a few additional jobs to add slightly more flexibility. In the VNS part of the algorithm, we incorporate diversification by moving jobs from $L[1:s]$ to $L[s+1:n]$ and/or vice versa. Roughly speaking, the VNS part selects the jobs to be serviced and the VND part reorganizes the serviced jobs.

In our implementation, the VND consists of the following 8 neighborhoods: Best Move, Swap, 4 different Block moves, Drop Bad, and Add Best Best. The VNS contains the following 11 neighborhoods: Add 5 random arcs, k-Block Add with $k \in 5, 2, 13, 17$, Gamma Drop, Add Random Randomly, Drop Random, Drop Best, New Start, Gamma Add Randomly. All neighborhoods are described in detail in [7]. We have performed extensive computational experiments with

many combinations of both these and other neighborhoods and concluded that this combination provides the best results for the smaller set of instances used for this analysis.

Additional experiments were also carried out where the ordering of the jobs in L was restricted to having jobs following each other that were close to each other or close to the depot. However these restrictions did not lead to significantly better solutions.

We use a skewed version of VNS where, in the first iteration, we only accept solutions that are strictly better than the current solution, whereas in subsequent iterations, given the current solution, x, we accept a new solution, x'' if it is strictly better than x or if $OBJ(x'') + \beta\rho(x, x'') > OBJ(x_B)$, where x_B is the best solution seen thus far. $\rho(x, x'')$ is a measure of the difference between the two input solutions and is defined as the number of jobs that are accepted in one of the solutions but not in the other. β measures the willingness to accept worsening solutions and is increased by $\frac{0.002|Q|x_B}{t_{\max}}$ in each iteration, where Q is the set of vehicles. Initially $\beta = 0$.

4.2.1. Splitting Procedure

Each time the order of the jobs in the list L is changed, the corresponding vehicle solution has to be re-calculated. In this section, we describe our splitting algorithm for doing this, which will be illustrated by an example.

The idea of optimally splitting a giant tour into several vehicle tours has been used in various settings. Prins [25] uses such a tour splitting algorithm for the vehicle routing problem, whereas Prins et al. [26] and Wøhlk [29] present similar algorithms for undirected arc routing where edge flipping is necessary. Prins et al. [27] provide a comprehensive survey of the use of splitting procedures in the literature.

A splitting algorithm where the number of vehicle routes is unlimited would apply to our problem as follows: Consider the order of the jobs given by $L[1 : n]$. We construct a network with $n+1$ nodes indexed by $\{0, \ldots, \mathrm{n}\}$. For any nodes $i \in \{0, n-1\}$ and $j > i$ we add an arc to the network with length $R(i, j)$ equal to the objective value (sum of prizes minus sum of travel times) of a single vehicle route servicing $L[i + 1 : j]$ in the order dictated by L. Because the order of the jobs is given, the value of $R(i, j)$ can easily be calculated for all i, j with $j > i$ resulting in a feasible route. If such a route is not feasible due to the time constraint, we set $R(i, j) = -\infty$.

An example of such a network is shown in Figure 2 for a problem with 10 jobs, where the numbers shown for each arc (i, j) are the values of $R(i, j)$. Only arcs with $R(i, j) > -\infty$ are shown.

A longest path from node 0 to any node s, in this network yields the optimal splitting of L along with the threshold s. In the example, the arcs of the longest

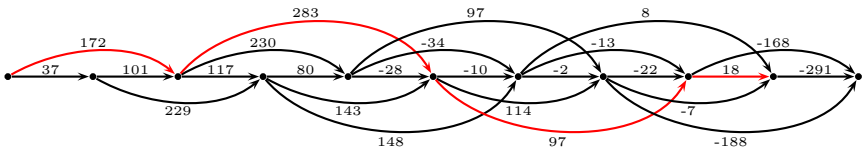

Figure 2: Example of tour splitting with unlimited vehicle routes.

path are shown in red. Here, it is optimal to service jobs 1 through 9 and reject job 10 and as result, the solution consists of four vehicle routes servicing $L[1:2], L[3:5], L[6:8]$, and $L[9]$, respectively and $s = 9$. This results in an objective value of 570.

The problem with using a traditional tour splitting algorithm, such as the one just described is that there is no control regarding the number of vehicles used. In the example, the optimal splitting used 4 vehicles and in the worst case, n vehicles could be used. To overcome these problems, we present here a state-space procedure which incorporates the limitations on the number of vehicles and the optional service. The state-space network corresponding to the example is shown in Figure 3, where we assume that only three vehicles are available.

We construct a network with $|Q| + 1$ by $n + 1$ nodes indexed by $\{v, j\}$ where $v \in \{0, \ldots, |Q|\}$ and $j \in \{0, \ldots, n\}$ as illustrated in Figure 3 for our example with 10 jobs and three vehicles. Here, $v = 0$ and $j = 0$ represents a dummy vehicle and dummy job, respectively.

For $v \in \{0, \ldots, |Q|-1\}$ and i, j with $j > i$, we add an arc from $\{v, i\}$ to $\{v+1, j\}$ if $R(i, j) > -\infty$ and the route servicing jobs $L[i + 1, j]$ can be performed by vehicle $v + 1$ while the solution respects the order of L. The latter implies that, since only the first vehicle can service $L[1]$, no arcs are leaving nodes $\{v, j\}$ with $j = 0$ and $v > 0$ or with $v = 0$ and $j > 0$. Furthermore, for $v > 0$ and $i > 0$, arcs from $\{v, i\}$ to $\{v+1, j\}$ are only included if the arc can be part of a feasible splitting of L.

A longest path from node $\{0, 0\}$ to any node $\{v', s\}$ in this network yields the optimal splitting of L along with the threshold s, where v' provides the number of vehicles used. In the example, the arcs of the longest path are shown in red. Here, it is optimal to service jobs 1 through 7 using three vehicles servicing $L[1 : 2], L[3 : 5]$, and $L[6 : 7]$, respectively and rejecting jobs $L[8 : 10]$. This results in $s = 7$ and an objective value of 569, which is less than the result where any number of vehicles could be used.

We use dynamic programming to find a longest path in this network. For this, we define $T(v, j)$ to be the objective value of servicing $L[1 : j]$ in that order, using v vehicles and let $T(v, j) = -\infty$ if it is not possible to service $L[1 : j]$ in

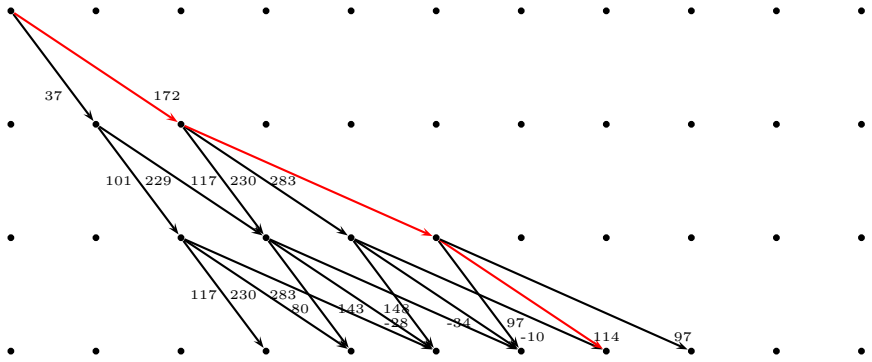

Figure 3: Example of state-space network.

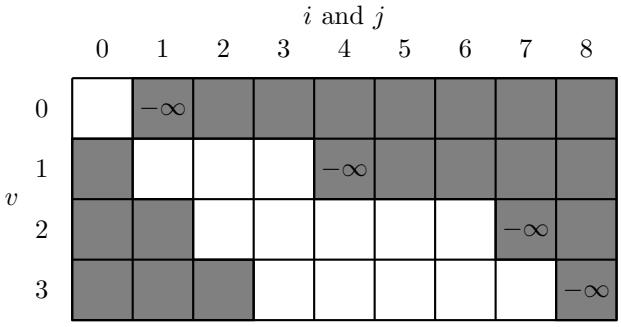

Figure 4: Table structure of the splitting algorithm.

that order using v vehicles. We seek the pair $v', s = \arg\max_{v,j} T(v, j)$ where v' is the number of vehicles used, and $L[1 : s]$ are the jobs to be serviced. Hence, job $s + 1$ is the first job not serviced. Our splitting procedure is shown in Algorithm 1 and Figure 4 shows the structure of the table, T, created by the algorithm. In Figure 4, the unshaded cells are those with $T(v, j) \neq -\infty$. The areas with $T(v, j) = -\infty$, determined by lines 4 and 16 in the algorithm are shown in the figure. All other shaded areas in Figure 4 have a *void* value and are not used in the algorithm.

5. Computational Experiments

5.1. Test Instances

We have generated a set of test instances for evaluation of the two procedures. The instances are based on those presented in [7], where we arbitrarily have

Algorithm 1 Split Procedure

```
1: function SPLIT
2:     Determine R(i, j)  ∀i, j with j > i
3:     T(0, 0) ← 0
4:     T(0, 1) ← −∞
5:     for v = 0 to |Q| − 1 do                          ▷ For each vehicle.
6:         for i = v to T(v, i) = −∞ do
7:             for j = i + 1 to R(i, j) = −∞ do
8:                 if T(v, i) + R(i, j) > T(v + 1, j) then
9:                     T(v + 1, j) = T(v, i) + R(i, j)
10:                end if
11:                if T(v + 1, j) > BestObj then
12:                    BestObj = T(v + 1, j)
13:                    s = j
14:                end if
15:            end for
16:            T(v + 1, j) = −∞                         ▷ First j not active in the for loop.
17:        end for
18:    end for
19: end function
```

used the third instance in each set to be extended to multiple vehicles. For the London instances, we have used both the third and the fifth instance in order to obtain a larger testbed. Table 1 gives some information on the data instances. All instances are available at http://www.optimization.dk/TD-MPARP.

Instance	RTT file	No. vehicles	No. prizes	Start time	Duration
A3	NW25	3, 5, 8	100	300	900
B3	NW25	3, 5, 8	100	300	900
C3	NW25	3, 5, 8	100	300	900
D3	NW25	3, 5, 8	100	300	900
E3	NW25	3, 5, 8	100	300	900
F3	NW100	3, 5, 8, 12	500	0	1440
London B3	london	3, 5, 8	75	360	600
London B5	london	3, 5, 8	75	360	600
London L3	london	3, 5, 8, 12	350	360	600
London L5	london	3, 5, 8, 12	350	360	600

Table 1: The test instances.

5.2. Results

The LANTIME algorithm was implemented on a PC with an Intel Core 2 Duo CPU, running at 2.2GHz with 2 GB of memory and the VNS algorithm was implemented on a PC with an Intel Core 2 Duo CPU, running at 2.40GHz and

Instance	VNS			LANTIME			Best
	Mean	Max	Min	Mean	Max	Min	known
mNW25-A3-3	5618.8	**5837.8**	5403.7	5553.3	5823.4	5451.2	6028.8
mNW25-A3-5	8038.2	**8215.5**	7858.1	8064.4	8201.8	7936.3	8588.2
mNW25-A3-8	10813.6	10889.4	10777.7	10872.0	**11110.9**	10588.7	11459.0
mNW25-B3-3	993.8	**1089.3***	942.3	845.9	923.8	735.0	1089.3
mNW25-B3-5	1103.2	**1130.8**	1085.1	927.9	1021.7	808.1	1181.7
mNW25-B3-8	1081.5	**1112.9**	1053.8	909.9	1012.5	819.9	1199.9
mNW25-C3-3	8240.7	8451.8	8130.5	8372.5	**8636.9**	8178.8	8939.7
mNW25-C3-5	11516.9	11796.1	11277.9	11699.2	**11962.8**	11486.8	12250.2
mNW25-C3-8	13574.0	13574.0	13574.0	13785.7	**13869.6**	13708.4	13951.1
mNW25-D3-3	4255.2	**4402.0**	4182.8	4205.9	4344.9	4134.0	4413.4
mNW25-D3-5	6608.5	6736.8	6475.1	6503.5	**6880.6**	6262.4	6993.2
mNW25-D3-8	10016.7	**10244.3**	9920.6	9597.0	9864.6	9406.9	10372.5
mNW25-E3-3	5011.8	5096.2	4893.9	5057.3	**5141.2**	4974.7	5317.8
mNW25-E3-5	7237.4	**7482.8**	7022.0	7206.1	7321.3	7015.0	7693.7
mNW25-E3-8	9553.9	9723.5	9327.6	9844.9	**9959.0**	9644.1	10235.6
mNW100-F3-3	10775.0	11180.8	10489.0	12651.0	**13237.9**	11625.3	14118.0
mNW100-F3-5	16459.3	16481.6	16456.9	18333.7	**20103.7**	17163.2	20553.0
mNW100-F3-8	22312.0	22312.0	22312.0	25815.6	**26605.4**	24463.5	28746.0
mNW100-F3-12	29608.3	29608.3	29608.3	34734.4	**35490.2**	34069.3	38489.4
mlondon-B3-3	2399.2	**2408.5**	2389.4	2368.8	2402.3	2342.8	2463.0
mlondon-B3-5	2450.4	2461.3	2442.8	2451.2	**2462.9**	2429.1	2512.4
mlondon-B3-8	2438.7	2443.4	2436.2	2420.1	**2454.6**	2397.0	2512.1
mlondon-B5-3	2473.4	2533.9	2357.3	2427.9	**2540.8**	2348.4	2583.8
mlondon-B5-5	2807.1	2826.4	2769.1	2849.5	**2879.7**	2823.7	3101.0
mlondon-B5-8	2853.0	2862.5	2848.5	2841.1	**2879.8**	2808.0	2975.5
mlondon-L3-3	3458.5	3905.7	3138.2	4547.5	**4790.8**	4283.1	5483.4
mlondon-L3-5	4457.2	4664.8	4307.8	5866.6	**6147.0**	5497.6	7141.4
mlondon-L3-8	5900.6	5900.6	5900.6	8096.0	**8476.9**	7601.8	9399.5
mlondon-L3-12	7636.0	7636.0	7636.0	10263.2	**10548.2**	10052.3	11326.1
mlondon-L5-3	3652.0	3915.2	3235.7	4465.4	**5047.7**	4169.2	5651.2
mlondon-L5-5	4308.4	4591.3	4067.9	6147.8	**6915.2**	5478.6	7590.4
mlondon-L5-8	5501.1	5501.1	5501.1	8480.8	**9133.7**	8128.8	9822.9
mlondon-L5-12	7613.3	7613.3	7613.3	10800.7	**11210.6**	10104.9	12105.4

Table 2: Results from 1 minute computing time

with 2.97GB of RAM. Although not identical, the two PCs have similar run time performance. The algorithms were both written in C++.

As for the single vehicle version of the problem [7], we run the two algorithms with a time limit of 1 min, 10 min, and 30 min. For each time limit, we solve each instance 10 times using different sets of random numbers within the algorithms. For each such 10 solutions, we report the value of the best solution (max), the

Instance	VNS			LANTIME			Best
	Mean	Max	Min	Mean	Max	Min	known
mNW25-A3-3	5853.5	**5924.0**	5729.9	5695.5	5823.4	5609.2	6028.8
mNW25-A3-5	8359.0	**8504.1**	8140.0	8223.4	8342.2	8125.0	8588.2
mNW25-A3-8	11216.1	**11336.3**	11102.9	11087.2	11214.1	10939.3	11459.0
mNW25-B3-3	1000.4	**1043.4**	939.6	856.7	923.8	798.7	1089.3
mNW25-B3-5	1137.8	**1169.0**	1095.2	955.8	1021.7	887.7	1181.7
mNW25-B3-8	1131.5	**1170.5**	1080.1	959.8	1012.5	909.6	1199.9
mNW25-C3-3	8685.9	**8939.7***	8531.9	8567.4	8636.9	8421.7	8939.7
mNW25-C3-5	12034.1	**12137.7**	11884.4	11867.7	12017.0	11730.6	12250.2
mNW25-C3-8	13884.9	**13938.6**	13832.5	13823.5	13900.4	13708.4	13951.1
mNW25-D3-3	4355.3	**4413.4***	4288.9	4310.7	4370.9	4233.1	4413.4
mNW25-D3-5	6790.5	**6883.0**	6712.0	6672.7	6880.6	6568.9	6993.2
mNW25-D3-8	10154.2	**10372.5***	9987.1	9897.0	10045.3	9594.4	10372.5
mNW25-E3-3	5231.1	**5294.1**	5152.4	5142.4	5230.1	5096.1	5317.8
mNW25-E3-5	7466.1	**7577.6**	7294.5	7349.7	7482.3	7280.4	7693.7
mNW25-E3-8	9946.3	10093.4	9813.6	10033.3	**10185.2**	9930.8	10235.6
mNW100-F3-3	12451.9	13072.8	11827.0	13344.0	**14118.0***	12883.6	14118.0
mNW100-F3-5	17873.4	18565.4	17105.4	19502.7	**20298.3**	18988.5	20553.0
mNW100-F3-8	25148.6	25345.9	24967.8	27546.1	**28259.4**	26816.3	28746.0
mNW100-F3-12	32673.1	32673.1	32673.1	37261.1	**38004.6**	36534.6	38489.4
mlondon-B3-3	2426.4	**2434.6**	2410.6	2385.7	2413.6	2367.3	2463.0
mlondon-B3-5	2501.2	**2512.4***	2476.3	2478.8	2489.9	2469.4	2512.4
mlondon-B3-8	2487.0	**2500.2**	2471.5	2468.7	2483.4	2457.4	2512.1
mlondon-B5-3	2555.5	**2583.8***	2506.3	2515.0	2545.3	2462.6	2583.8
mlondon-B5-5	2909.1	**3101.0***	2842.4	2881.5	2897.6	2867.6	3101.0
mlondon-B5-8	2916.5	**2948.0**	2888.2	2912.4	2928.0	2891.8	2975.5
mlondon-L3-3	4352.0	4667.9	4126.0	5231.9	**5362.0**	5103.6	5483.4
mlondon-L3-5	5483.5	5780.4	5111.6	6814.0	**7128.6**	6512.5	7141.4
mlondon-L3-8	7339.2	7595.2	7183.2	8953.1	**9120.4**	8764.6	9399.5
mlondon-L3-12	8733.5	8733.5	8733.5	11007.1	**11241.3**	10709.9	11326.1
mlondon-L5-3	4399.0	4578.7	4149.0	5427.2	**5650.0**	5254.9	5651.2
mlondon-L5-5	5556.0	5686.6	5430.7	7127.7	**7398.5**	6652.5	7590.4
mlondon-L5-8	7205.4	7506.1	6726.1	9236.0	**9541.2**	8887.1	9822.9
mlondon-L5-12	9100.4	9100.4	9100.4	11515.1	**11808.4**	11138.6	12105.4

Table 3: Results from 10 minute computing time

worst solution (min), and the mean of the 10 solutions.

The results are reported in tables 2 through 4, where the last column in each table reports the best value obtained during any of the runs. This best known value is often obtained in the 30 min runs. We indicate by bold font the best solution obtained by one of the algorithms for the given run time. We use an asterisk to indicate when the best solution coincides with the best known

Instance	VNS			LANTIME			Best
	Mean	Max	Min	Mean	Max	Min	known
mNW25-A3-3	5921.3	**6028.8***	5801.9	5785.7	5860.3	5692.2	6028.8
mNW25-A3-5	8490.6	**8588.2***	8345.4	8329.6	8506.9	8189.5	8588.2
mNW25-A3-8	11262.7	**11459.0***	11076.7	11201.7	11405.4	11073.0	11459.0
mNW25-B3-3	1007.2	**1030.3**	967.6	904.3	964.2	801.7	1089.3
mNW25-B3-5	1160.0	**1181.7***	1138.5	959.4	1021.7	905.8	1181.7
mNW25-B3-8	1162.1	**1199.9***	1130.8	974.0	1012.5	921.5	1199.9
mNW25-C3-3	8774.6	8898.5	8588.9	8672.6	**8912.1**	8535.3	8939.7
mNW25-C3-5	12109.8	**12250.2***	11975.9	11935.9	12017.0	11821.8	12250.2
mNW25-C3-8	13905.3	**13951.1***	13869.9	13837.3	13903.0	13708.4	13951.1
mNW25-D3-3	4225.2	4357.0	4021.0	4354.3	**4408.0**	4305.0	4413.4
mNW25-D3-5	6882.0	**6993.2***	6797.6	6754.3	6880.6	6663.5	6993.2
mNW25-D3-8	10189.5	**10290.0**	10043.1	9992.5	10089.2	9828.4	10372.5
mNW25-E3-3	5239.8	**5317.8***	4964.5	5174.0	5279.3	5105.7	5317.8
mNW25-E3-5	7541.0	**7693.7***	7440.1	7417.6	7532.5	7343.4	7693.7
mNW25-E3-8	10165.8	**10235.6***	10104.2	10076.6	10185.2	10009.5	10235.6
mNW100-F3-3	13377.9	13711.9	12858.2	13681.8	**14118.0***	13681.8	14118.0
mNW100-F3-5	19000.0	19558.4	18663.0	20050.1	**20553.0***	20050.1	20553.0
mNW100-F3-8	25957.9	26192.8	25521.2	28339.7	**28746.0***	28339.7	28746.0
mNW100-F3-12	34505.2	34771.2	34317.2	38101.0	**38489.4***	38101.0	38489.4
mlondon-B3-3	2429.2	**2463.0***	2404.9	2396.9	2419.3	2373.1	2463.0
mlondon-B3-5	2503.2	**2511.7**	2486.9	2482.9	2491.4	2475.5	2512.4
mlondon-B3-8	2499.2	**2512.1***	2484.1	2476.9	2483.4	2469.0	2512.1
mlondon-B5-3	2553.1	**2580.2**	2466.5	2527.8	2545.3	2497.9	2583.8
mlondon-B5-5	2890.5	**2937.8**	2842.9	2897.3	2921.2	2880.0	3101.0
mlondon-B5-8	2937.3	**2975.5***	2902.5	2923.3	2943.6	2912.8	2975.5
mlondon-L3-3	4568.2	4965.1	4163.7	5359.3	**5483.4***	5236.2	5483.4
mlondon-L3-5	5928.7	6193.3	5685.1	7065.6	**7141.4***	6880.4	7141.4
mlondon-L3-8	7744.8	7940.3	7548.4	9234.0	**9399.5***	9062.7	9399.5
mlondon-L3-12	9939.7	9939.7	9939.7	11229.4	**11326.1***	11097.6	11326.1
mlondon-L5-3	4714.3	4972.4	4408.8	5562.2	**5651.2***	5425.5	5651.2
mlondon-L5-5	6083.7	6323.1	5857.9	7492.2	**7590.4***	7336.6	7590.4
mlondon-L5-8	7791.7	8039.4	7578.9	9582.8	**9822.9***	9374.7	9822.9
mlondon-L5-12	10189.0	10189.0	10189.0	11926.6	**12105.4***	11742.7	12105.4

Table 4: Results from 30 minute computing time

solution for that instance.

Table 5 summarizes the results. Here, we first state the number of times each algorithm obtained the best solution within the given run time. Next, we state the number of times the best solution for each run time coincides with the best known solution. In the second part of the table, we indicate the average, minimum and maximum percentage variation of the solution profit for the 10

	VNS			LANTIME		
	1 min	10 min	30 min	1 min	10 min	30 min
Best solution	9	20	19	24	13	14
Best known obtained	1	6	14	0	1	12
Average variation	4.6	4.6	4.3	8.6	4.9	3.8
Minimum variation	6.6	0.0	0.0	1.2	0.8	0.6
Maximum variation	22.2	12.5	17.5	23.4	14.6	18.0

Table 5: Summary of the results

runs for each instance, where precentage variation is calculated as follows:

$$\text{Percentage instance variation} = \frac{\text{max value} - \text{min value}}{\text{average value}} \cdot 100$$

Hence, these numbers report the amount of variation in the results obtained.

It is interesting to observe that for VNS, the minimum variation is sometimes zero. This indicates that for at least one instance, all 10 runs have provided the same solution value. One example of this behavior is found for the 10-minute runs of instance mNW100-F3-12, where the VNS algorithm gets caught in a local minimum. The average variation for VNS is relatively unaffected by an increase in the run time, whereas the table shows that for LANTIME, the average variation decreases when the run time increases. This indicates that LANTIME converges to solutions of similar quality with increased run time.

We consider the relative performance of the two algorithms when the run time is increased by comparing across Tables 2 through 4. For the 1-minute runs, the LANTIME algorithm often outperforms VNS, but when a larger run time is provided, the balance changes in favor of VNS.

Furthermore, for the three sets of large instances, F3, London L3, and London L5, the LANTIME algorithm performs significantly better than the VNS irrespectively of the time available.

From this analysis, we conclude that if only short run time is available, the LANTIME algorithm is the best choice. If longer run time is available, the VNS should be used, unless the instance is large, in which case LANTIME is better.

The latter part of this conclusion can be explained by the structure of the solutions in the algorithms. As explained in Section 4.1, in the implementation of the LANTIME algorithm, each vehicle route is maintained and modified by the neighborhoods. In the VNS, on the other hand (see Section 4.2), a giant tour is maintained, and every time a neighbor solution needs to be evaluated, the splitting procedure is executed to partition the tour into vehicle routes. The run time of this splitting procedure increases with the instance size, and in particular with the number of vehicles available. As a result, significantly

fewer solutions can be evaluated per time unit and as a result, fewer iterations. Indeed, for the largest instances, when providing only one minute, the VND algorithm reached its time limit before it finished its search from the first call of the VNS, and therefore the algorithm terminated prematurely.

5.3. Results of Increasing the Number of Vehicles

Figure 5 shows how the average solution profit increases as vehicles are added to a problem instance. The general shape of this curve is as expected. It begins with a sharp increase as the highly profitable arcs can be added to the route of the extra vehicle. This increase slows, at around 4 vehicles, as only less profitable arcs remain. Eventually the curve reaches a plateau after all profitable arcs have been included in the solutions found. Any extra vehicles beyond this point are superfluous.

One interesting issue that can be observed is that the average performance of LANTIME degrades after the plateau. This is due to the fact that increasing the number of vehicles will increase the number of possible neighborhood moves and hence the complexity of the search neighborhood. As neighborhood complexity increases, a meta-heuristic, such as LANTIME, that does not exhaustively search each current neighborhood will not always make particularly good neighborhood moves. Figure 6 shows the best and worst case solution profits for both meta-heuristics. The poor average performance of LANTIME can be explained by the increasingly poor worst case performance. That said, the best case performance of LANTIME remains better than that for the VNS meta-heuristic. For problem instances with increasingly complex neighborhoods VNS rapidly converges to a local optimum. This optimum is a good solution due to the exhaustive nature of the neighborhood search; however, the lack of variance means that as the solution sample size increases LANTIME will be more likely to find a more profitable solution.

6. Concluding Remarks

This paper has investigated a new problem in logistics and scheduling referred to as the TD-MPARP. The TD-MPARP is related to other Prize-Collecting Arc Routing Problems, but is different because it takes into account time-dependent travel times between locations. This feature makes the model harder to solve than problems where travel times are assumed to be static, but it also makes the model more realistic, especially when applied in urban areas where traffic congestion can lead to significant differences in journey times at different times of day.

Literature related to the TD-MPARP has been reviewed and classified, particularly noting the differences between *efficiency oriented* and *overloaded* problems. The TD-MARP can be regarded as an example of an *overloaded* problem.

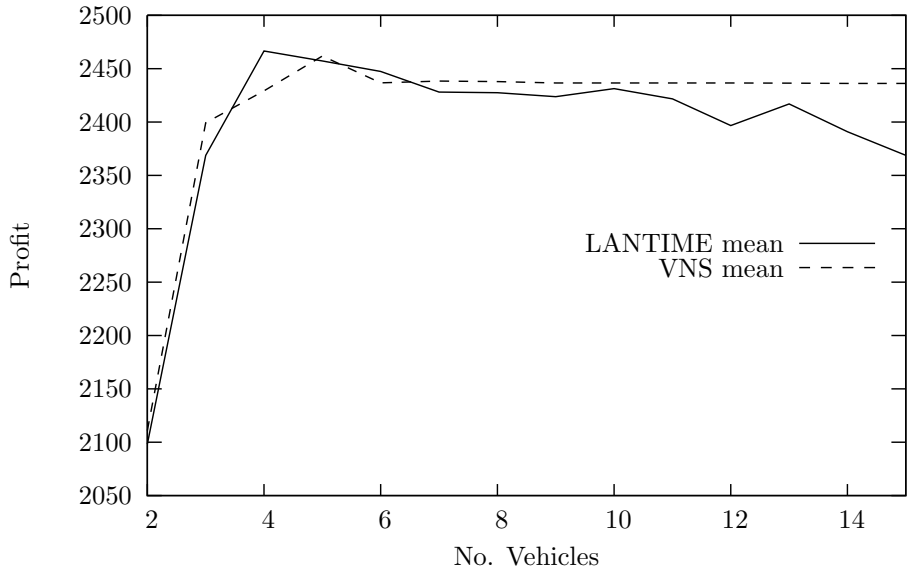

Figure 5: The effect of increasing the number of vehicles on profit for the mlondon-B3 instance (run time 1 minute; sample size 10).

A precise mathematical formulation of the TD-MPARP has been produced. However the formulation is highly non-linear and it would not be practical to find exact optimal solutions using this formulation for problems of the size used in this study. Two different styles of heuristic algorithms have been developed to solve a set of benchmark instances.

Firstly, a tabu search based algorithm, named LANTIME, was modified so that it could be applied to the TD-MPARP. Secondly, a new Variable Neighborhood Search (VNS) algorithm was introduced for the TD-MPARP, based on an algorithm developed by the authors for the TD-PARP, but extended to deal with multiple vehicles.

The algorithms were tested over a range of different run times and the results show that both algorithms are capable of providing good solutions. One algorithm does not dominate the other under all conditions. Using the longest run times of 30 minutes per instance, VNS tended to perform better on the smaller instances, while LANTIME performed better on the larger instances.

Further experiments compare the results from the algorithms as the number of vehicles increases. The best result from LANTIME is generally better than the best result from VNS over a set of 10 runs. However, for an individual instance, LANTIME exhibits a relatively poorer average performance as the number of vehicles increases compared to the VNS algorithm.

The results show that good solutions can be found for the TD-MPARP for

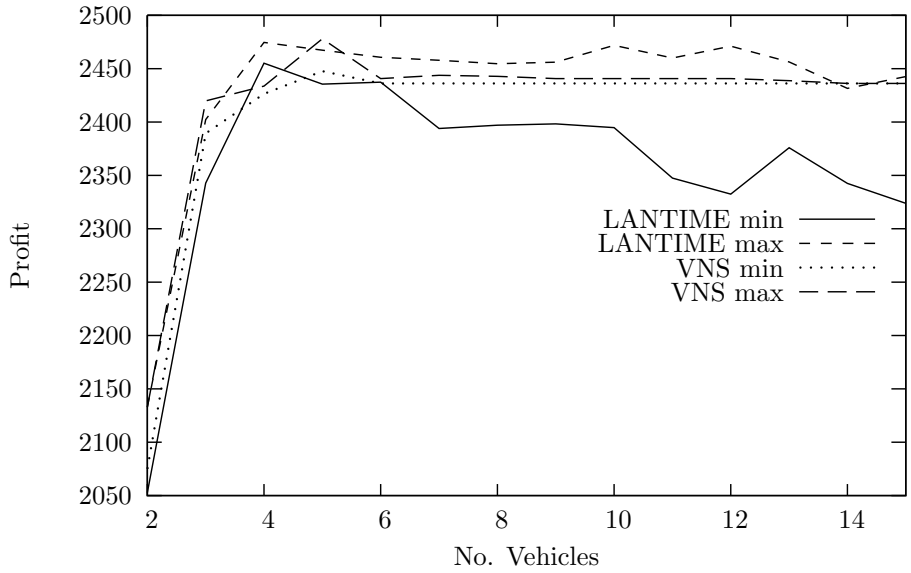

Figure 6: The effect of increasing the number of vehicles on profit for the mlondon-B3 instance (run time 1 minute; sample size 10).

problems of a size that might be encountered in practice. The incorporation of time-dependent journey times enables situations where varying levels of traffic congestion are significant to be modelled more accurately

7. References

References

[1] Julián Aráoz, Elena Fernández, and Carles Franquesa. The Clustered Prize-Collecting Arc Routing Problem. *Transportation Science*, 43(3):287–300, 2009.

[2] Julián Aráoz, Elena Fernández, and Oscar Meza. Solving the Prize-Collecting Rural Postman Problem. *European Journal of Operational Research*, 196:886–896, 2009.

[3] Julián Aráoz, Elena Fernández, and Cristina Zoltan. Privatized Rural Postman Problems. *Computers and Operations Research*, 33:3432–3449, 2006.

[4] Claudia Archetti, Dominique Feillet, Alain Hertz, and M. Grazia Speranza. The Undirected Capacitated Arc Routing Problem with Profits. *Computers and Operations Research*, 37:1860–1869, 2010.

[5] Claudia Archetti, M. Grazia Speranza, Ángel Corberán, José M. Sanchis, and Isaac Plana. The team orienteering arc routing problem. *Transportation Science. In press.*, September 2013.

[6] Sundararajan Arunapuram, Kamlesh Mathur, and Daniel Solow. Vehicle Routing and Scheduling with Full Truckloads. *Transportation Science*, 37(2):170–182, 2003.

[7] Dan Black, Richard Eglese, and Sanne Wøhlk. The time-dependent prize-collecting arc routing problem. *Computers and Operations Research*, 40(2):526–535, 2013.

[8] Ángel Corberán, Elena Fernández, Carles Franquesa, and José M. Sanchis. The Windy Clustered Prize-Collecting Arc Routing Problem. *Transportation Science*, 45(3):317–334, 2011.

[9] S. Dabia, S. Ropke, T. van Woensel, and T. De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396, November 2012.

[10] Mauro Dell'Amico, Francesco Maffioli, and Peter Vrbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308, July 1995.

[11] Jacques Desrosiers, Gilbert Laporte, Michel Sauve, Francois Soumis, and Serge Taillefer. Vehicle Routing with Full Loads. *Computers and Operations Research*, 15(3):219–226, 1988.

[12] Richard Eglese, Will Maden, and Alan Slater. A Road Timetable$^{\text{TM}}$ to aid Vehicle Routing and Scheduling. *Computers and Operations Research*, 33:3508–3519, 2006.

[13] Jalel Euchi, Habib Chabchoub, and Adnan Yassine. Metaheuristics Optimization via Memory to Solve the Profitable Arc Tour Problem. 8^{th} *International Conference of Modeling and Simulation*, 2010.

[14] Dominique Feillet, Pierre Dejax, and Michel Gendreau. The Profitable Arc Tour Problem: Solution with a Branch-and-Price Algorithm. *Transportation Science*, 39(4):539–552, 2005.

[15] Manfred Gronalt, Richard F. Hartl, and Marc Reimann. New Savings Based Algorithm for Time Constrained Pickup and Delivery of Full Truckloads. *European Journal of Operational Research*, 151:520–535, 2003.

[16] Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable Neighborhood Search. *European Journal of Operational Research*, 191(3):593–595, 2008.

[17] Kieran G. Harwood, Christine L. Mumford, and Richard Eglese. Investigating the Use of Metaheuristics for Solving Single Vehicle Routing Problems with Time-Varying Traversal Costs. *Journal of the Operational Research Society*, 64(1):34–47, 2013.

[18] S. Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle Dispatching with Time-Dependent Travel Times. *European Journal of Operational Research*, 144:379–396, 2003.

[19] Ran Liu, Zhibin Jiang, Richard Y.K. Fung, Feng Chen, and Xiao Liu. Two-phase heuristic algorithms for full truckloads multi-depot capacitated vehicle routing problem in carrier collaboration. *Computers & Operations Research*, 37(5):950–959, May 2010.

[20] Will Maden, Richard Eglese, and Dan Black. Vehicle Routing and Scheduling with Time-Varying Data: A Case Study. *Journal of the Operational Research Society*, 61:515–522, 2010.

[21] Chryssi Malandraki and Mark S. Daskin. The Maximum Benefit Chinese Postman Problem and the Maximum Benefit Traveling Salesman Problem. *European Journal of Operational Research*, 65:218–234, 1993.

[22] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.

[23] Jenny Nossack and Erwin Pesch. A truck scheduling problem arising in intermodal container transportation. *European Journal of Operational Research*, 230(3):666–680, November 2013.

[24] W.L. Pearn and K.H. Wang. On the Maximum Benefit Chinese Postman Problem. *Omega*, 31:269–273, 2003.

[25] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31:1985–2002, 2004.

[26] Christian Prins, Nacima Labadi, and Mohamed Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47:507–536, 2009.

[27] Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-First Split-Second Methods for Vehicle Routing Problems: A review. *Transportation Research Part C*, 40:179–200, 2014.

[28] David Soler, José Albiach, and Eulalia Martínez. A way to optimally solve a time-dependent Vehicle Routing Problem with time windows. *Operations Research Letters*, 37:37–42, 2009.

[29] Sanne Wøhlk. An approximation algorithm for the Capacitated Arc Routing Problem. *The Open Operational Research Journal*, 2:8–12, 2008.

[30] E.E. Zachariadis and C.T. Kiranoudis. Local Search for the Undirected Capacitated Arc Routing Problem with Profits. *European Journal of Operational Research*, 210:358–367, 2010.