WORKING PAPER L-2008-09

WP

Ellis L. Johnson & Sanne Wøhlk

Solving the Capacitated Arc Routing Problem with Time Windows using Column Generation

CORAL – Centre for Operations Research Applications in Logistics

Solving the Capacitated Arc Routing Problem with Time Windows using Column Generation

Ellis L. Johnson* Sanne Wøhlk†

January 26, 2009

Abstract

In this paper we consider the Capacitated Arc Routing Problem with Time windows. We suggest two algorithms for solving the problem to optimality and a heuristic for obtaining high quality solutions. To our knowledge this is the first paper to consider optimal solution of that problem.

keywordsCARP-TW, column generation, heuristics

1 Introduction

In this paper, we consider the solution of the Capacitated Arc Routing Problem with Time Windows (CARP-TW). It is the problem of servicing a set of demand edges in a graph using a fleet of vehicles, with the restriction that the service of each required edge must begin within a pre-specified time slot. The problem is an extension of the classical Capacitated Arc Routing Problem (CARP).

We suggest a strategy for solving the CARP-TW to optimality, which is based on the generation of feasible vehicle tours and the solution of a large Set Partitioning Problem. We present two methods for solving the Set Partitioning Problem: A two-phase method and an iterative method. When time windows are tight it is possible to solve the CARP-TW to optimality using this strategy, as the computational results show. When time windows are wide or the instances are large, solving the problem to optimality is extremely hard, which is also the case for the classical CARP. We therefor suggest a heuristic procedure, where the idea is to generate only tours that are good in some pre-specified sense and subsequently use the presented methods for solving a Set Partitioning Problem based on these columns. As can be seen from the computational results, this heuristic often finds solutions within few percent of the optimal solution.

To our knowledge, no attempts have been made to solve the CARP-TW by exact methods. In the next section, we will describe our algorithm for doing so. In Section 3, we show how this procedure can be used in a heuristic for the CARP-TW, and in Section 4, we present our computational results and offer our conclusion.

1.1 Related Literature

CARP-TW has not been studied much in the literature, but does arise in some of the applications of arc routing. Street Sweeping as presented by Bodin and Kursh in [8] is a practical application of arc routing, where each street has to be swept within a specified time window. Routing of Winter Gritters are considered by Eglese in [15],

*School of Industrial and Systems Engineering, Georgia Institute of Technology
†Aarhus School of Business, Århus, Denmark

where some streets must be serviced within two hours, other within four etc. This problem can clearly be considered as a CARP-TW, where the time windows are rather wide. Flight legs in Airline Scheduling [10, 16, 27] have fixed departure time and can therefore be considered as having time windows of zero/short length, resulting in an extra complicated version of the directed CARP-TW. Finally, theoretical aspects of the CARP-TW are considered by Wøhlk in [26] and Mullaserin considers a directed version of the problem in [24]. In [21], Labadi et al. presents a GRASP algorithm for the CARP-TW.

Algorithms have been constructed for solving the classical CARP to optimality. In [18], Hirabayashi suggests a Branch and Bound based algorithm. Both Baldacci and Maniezzo [3] and Aragão et al. [14] propose an exact solution strategy for the CARP which is based on transforming the problem to a corresponding node routing problem (VRP), which is in turn solved by state-of-the-art algorithms. Branch-Price-and-Cut algorithms for solving the CARP to optimality without transformation to an equivalent node routing problem is presented by Belenguer et al. in [7] and by Letchford and Oukil in [23]. Furthermore, valid inequalities and separation routines for these are presented by Belenguer and Benavent in [5] and by Letchford in [22]. Finally, Belenguer and Benavent presents a cutting plane algorithm for the CARP in [6].

1.2 Problem Definition

In the CARP-TW, we are given an undirected connected graph $G(N, E, C, Q)$, where N is the set of nodes, E is the set of edges, C is a cost matrix with $c_{ij} \geq 0 \ \forall \ (i,j) \in E$, and Q is a demand matrix where $q_{ij} \geq 0$ is the demand of the edge $(i,j) \in E$. If $q_{ij} > 0$, the corresponding edge is said to be required. Each required edge has a time window $[a_{ij}, b_{ij}]$ within which the service of that edge must begin. For each pair of nodes, i and j, $SPL(i,j)$ denotes the cost of a shortest path from i to j and t_{ij} denotes the travel time from i to j. It is assumed that travel time is proportional to cost, depending on wether the edge is serviced or not, i.e.

$$t_{ij} = \begin{cases} \Psi c_{ij} & \text{when } (i,j) \text{ is serviced} \\ \Phi c_{ij} & \text{when } (i,j) \text{ is not serviced} \end{cases}$$

where $\Psi, \Phi \geq 0$ are constants. This assumption is realistic when considering the applications of the problem. Without this assumption, the column generation procedure of section 2.1 must be changed accordingly and the remaining solution procedure can be kept unchanged.

Node 1 is a special depot node in which a fleet of K identical vehicles, each with capacity W is stationed. The CARP-TW amounts to finding tours for the vehicles in such a way that 1) Each required edge is serviced by exactly one vehicle, 2) The service of each required edge starts within the time window of that edge, 3) The sum of the demands of those edges serviced by each vehicle does not exceed W, and 4) The total cost of the tours is minimized.

It is possible to transform an instance of the CARP-TW to an instance of the VRP-TW. The procedure for doing this is described in Wøhlk [26] and is similar to the procedure for transforming the classical CARP to VRP presented in Assad et al. [2].

2 Solution Procedure

A first attempt to solve a problem to optimality would be to start out with an ILP formulation of the problem, and run a Branch and Bound algorithm based on that model. Unfortunately, for many problems, this is not possible because it requires an

enormous amount of computer power and time. This is especially true for routing problems, as such models are huge and very sparse, i.e. the number of non zeros is small compared to the number of zeros. Even worse, these models are highly symmetric.

Therefore we turn our attention to the related Airline Scheduling Problem. The solution strategy that has proved successful for the Airline Scheduling Problem, and also for many other problems, is that of Column Generation. See for example [1, 9, 17]. The idea in is that of decomposing the problem into smaller parts, generally referred to as the Master Problem and the Sub Problem, such that each of these two problems is easier to solve than the original problem. For Airline Scheduling Problems this is a matter of generating legal pairings, and then use these pairings to construct an optimal flight plan. The pairings, which will then become columns in a Set Partitioning Problem, can either be generated continuously as in a Dantzig-Wolfe decomposition setup [12, 13, 4] or up front as is done by Anbil et al. for the Crew-Pairing Problem in [1].

Solving the Subproblem is often very complicated, in particular the part where one needs to prove that no column of negative reduced cost exists. Furthermore, if cutting is used within the Branch and Bound procedure, the structure of the subproblem can be destroyed. Due to the similarity of the problems, we have chosen to solve the CARP-TW by Column Generation with columns generated up front, since this has proved to be a successful strategy for the Airline Scheduling Problem.

A decomposition of the CARP-TW is to let the Sub Problem be that of generating vehicle tours that are feasible with respect to both capacity and time windows. Thereby, the columns in the Master Problem correspond to feasible vehicle tours. Let E_R be the set of required edges and let $\mathcal{R}$ be the set of all feasible routes. The cost of route $j \in \mathcal{R}$ is denoted c_j and α_{ej} is a parameter of value 1 if edge e is serviced on route j and 0 otherwise. Finally, let x_j be a binary variable taking the value 1 if route j is included in the selected route collection and 0 otherwise. This leads to the following Set Partitioning formulation of the CARP-TW:

$$
\begin{aligned}
\min \quad & \sum_{j \in \mathcal{R}} c_j x_j \\
s.t. \quad & \sum_{j \in \mathcal{R}} \alpha_{ej} x_j = 1 \quad \forall e \in E_R \\
& x_j \in \{0,1\} \quad \forall j \in \mathcal{R}
\end{aligned}
$$

In the following we describe our procedure for solving the CARP-TW using this formulation. In Section 2.1, we explain how the set $\mathcal{R}$ of feasible routes is generated, and in Section 2.2, we explain the details of solving the LP-relaxation of the problem. The optimal solution of this relaxed problem gives us the initial basis for the ILP. In Sections 2.3, and 2.4, we show two strategies for solving the problem to integer optimality.

2.1 Generation of Columns

To generate all feasible columns for the problem, we use a depth-first search on the required edges. The search path is broken when no required edge can be added. When time windows are tight, the number of non-zeros in each column is small. Often as small as 4 or 5. This means that even though a depth-first search tree is rather wide, the depth is small, and thereby the time for generating the columns is relatively small. Clearly, the depth of the search path, and thereby the number of non zeros, increases as the time windows become wider.

When using depth-first search to generate columns, we get the order of the edges serviced on a tour, e.g. e_1, e_2, e_4. Since the number of edges serviced on a tour is

relatively small, we can quickly get the minimum cost route through the edges, which is feasible with respect to the time windows. Using this method, we get between 61000 and 8.9 million columns, depending on the instance. See Table 1 for details.

If time windows are relatively wide, the search will encounter several tours servicing the same set of edges, e.g. e_1, e_2, e_4 and e_4, e_1, e_2. Clearly, at most one of these tours can be present in an optimal solution. By removing such duplicates and only keeping the one with smallest cost, the size of the largest instance can be reduced to approximately 3.9 million columns.

It should be pointed out that the methods described in the following sections are superior to that of solving the complete Set Partitioning Problem with a standard Branch and Cut, both regarding speed and regarding the ability to solve large instances, and the implementation time is very limited. As a consequence, these methods may also prove useful in other situations, where a Set Partitioning Problem of the same kind arises.

2.2 Initial Basis

The next step in our solution procedure is to solve the LP-relaxation of the Set Partitioning Problem. Doing this is interesting for two reasons. First, the optimal value of the LP-relaxation is a lower bound on the optimal value of the problem. Second, we use the optimal basis of the LP-relaxation to form part of the initial set of columns for the ILP solution process.

Due to the number of columns in the problem, it is not possible to solve the LP-relaxation directly using the simplex algorithm. We have chosen to use a Subproblem Method as suggested by [1]. Here we will first describe the idea in the method, and then discuss how the core method can be improved as described in [19] to get a Primal-Dual Subproblem Simplex.

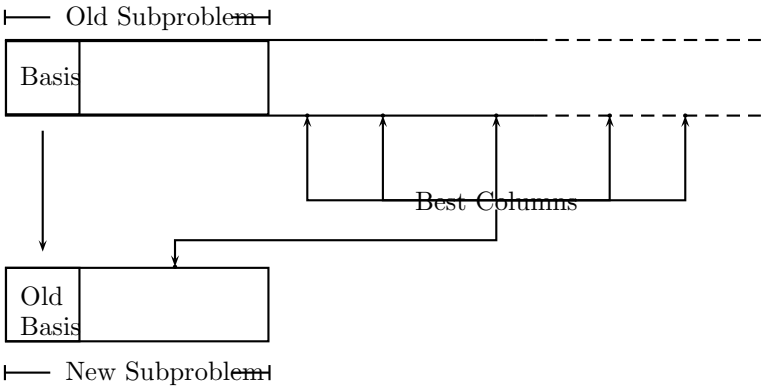

Figure 1: The Primal Subproblem Method.

The Primal Subproblem Method is illustrated in Figure 1. The idea is as follows: Initially a set of columns is chosen to form the first subproblem. For simplicity, this is chosen such that a feasible solution exists. The number of columns to be included in the subproblem depends on the size of the problem. The subproblem is solved to optimality using a standard simplex algorithm. Using the dual variables from this subproblem solution, the reduced costs are calculated for all columns. All columns that are not in the optimal basis are removed from the subproblem, since these all have non-negative reduced cost, and the columns with the most negative reduced cost are included in the subproblem instead. The number of columns to be included is kept small enough for the subproblem to be solvable reasonably fast. The process

is repeated with this new subproblem until no column with negative reduced cost can be found. Since we have generated all feasible columns up front, it is known that this solution is optimal for the whole problem.

It is experienced by Anbil et al. in [1] that this method is faster than a general simplex for two main reasons. First of all, the complete set of columns is only inspected, when the current subproblem is solved to optimality. It is experienced by [1] that only about 25 subproblems are solved, and hence this is only the number of times that we calculate reduced costs for all columns. Second, the subproblem is kept small, resulting in the solution of the subproblem being fast. It is our experience with the more involved Primal-Dual Subproblem method, that only 5-10 subproblems need to be solved. To speed up the selection of columns to be included in the subproblem, we use a threshold for the size of the reduced cost. If the number of columns to be included is too small, we iterate with these columns, but decrease the absolute value of the threshold for the next iteration. If the number of columns to be included is too large, only a fixed number of columns are added to the subproblem in order to keep the size down and the absolute value of the threshold is increased for the next iteration.

A method for speeding up the Primal Subproblem Simplex by turning it into a Primal-Dual Subproblem Simplex is presented in [19]. The method is further explained in [20], where a parallelization of the algorithm is also given. We have used this algorithm to solve the LP-relaxation of the Set Partitioning Problem for the CARP-TW, with the number of columns stabilized as described above.

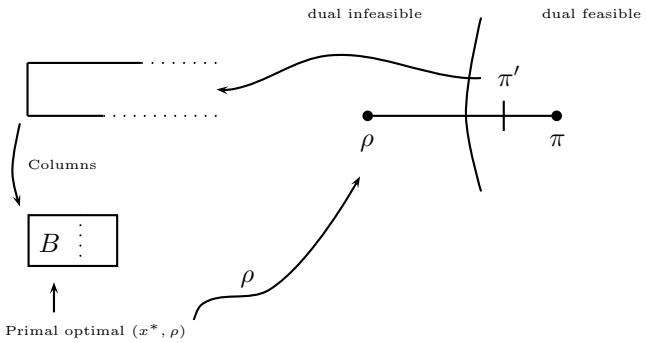

Figure 2: The Primal-Dual Subproblem Simplex.

The idea in the Primal-Dual Subproblem Simplex, which is illustrated in Figure 2, is the following: As in the Primal Subproblem method, we start with a primal feasible solution. For the Primal-Dual method, we also need a dual feasible solution, π. As in the Primal Subproblem method, we solve the subproblem, which consists of a subset of the columns, to optimality. Let the primal optimal solution be x^*, and let the corresponding vector of dual variables be ρ. In the primal method, we use ρ to calculate reduced costs for all the columns and choose columns with negative reduced cost to enter the new subproblem. Here, the idea is to use ρ to improve π, and then use this new π to calculate reduced costs, and determine which columns to add to the new subproblem. For use in the further solution process, the columns are marked when they enter the subproblem. It is known that if ρ is dual feasible for the whole problem, the current x^* is optimal. If this is not the case, ρ is dual infeasible, and π is dual feasible, where the latter is always true. A new dual feasible solution is now constructed as a convex combination of ρ and π, i.e.

$$\pi' = t\pi + (1-t)\rho$$

where $0 \leq t \leq 1$. In order to get as close to primal optimality as possible, t should be chosen in such a way that $\pi' b$ is as large as possible, without violating dual feasibility. As in [19], this is done by setting

$$t = \max_{j:\overline{c}_j^\rho < 0} \{0, \frac{-\overline{c}_j^\rho}{\overline{c}_j^\pi - \overline{c}_j^\rho}\}$$

where $\overline{c}_j^\rho$ is the reduced cost for the j'th column, calculated by using ρ as dual variable, and similarity for $\overline{c}_j^\pi$. The algorithm iterates as the Primal Subproblem algorithm, and stops when ρ becomes dual feasible.

2.3 Solving the ILP using a Two-Phase Method

Only the smallest instances of the Set Partitioning Problem could be solved by a standard CPLEX 8.1 Branch and Cut algorithm. In this section we describe the first of our methods for solving the ILP to optimality.

The problems that we need to solve have few rows and a huge number of columns (millions). An optimal integer solution is made up by a small set of columns, and all other columns are unused. Therefore, the idea is to select a set of columns that is likely to contain the optimal set. This set of columns should be small enough for the ILP consisting of these columns to be solvable within reasonable time, but large enough for the columns needed for the optimal solution to most likely be included in the set. Solving this ILP should give us the optimal solution, or, in case the columns needed for the optimal solution are not in the set, a good solution. In the second phase of the method, an ILP is constructed with the whole set of columns, and the solution is initiated with the (near) optimal integer solution from the first phase. Even though this ILP is too large to be solvable under normal conditions, it turns out that by this initiation, the algorithm can limit the search tree by cutting off parts that cannot contain a better solution. In both phases, the ILP is solved by CPLEX 8.1 Branch and Cut. If the optimal solution is found in the first phase, the second phase only needs to prove optimality of this solution. Below we will give further details of the two phases.

Clearly, the main issue in this solution method is the selection of columns to be included in the small ILP. Three things must be taken into consideration in this selection process. 1) We must make sure that a feasible integer solution exists among the selected columns, 2) The number of columns should be rather small, and 3) We want it to be likely that the columns are in the optimal solution.

The first issue is easy to overcome. We simply add all unit-columns to the problem, i.e. all columns that correspond to a tour servicing a single required edge. Since such a tour is legal for every required edge, this set of columns ensures integer feasibility. To overcome the latter two issues, we make the following observation: During the optimization of the LP-relaxation, columns that appeared to be good were at some point added to the subproblem. On the other hand, all columns that have been in the subproblem at some point after the initial iteration seemed good at that time. These columns were marked during the solution of the LP-relaxation. When preparing for the small ILP, the columns are chosen among the marked ones. If few columns are marked, they are all included in the small ILP, otherwise we randomly choose among the marked columns to obtain the desired number of columns in the ILP problem. Furthermore, the optimal basis from the LP-relaxation is included in the small ILP and is used as start basis.

After solving the small ILP, all columns are added to the problem, and the ILP is solved to integer optimality. To ensure that this process is relatively fast, the solution procedure is initiated with the optimal solution of the small ILP. That way, large parts of the search tree can be cut-off, and as a result, the computation time

is reduced. Clearly, this is much faster than solving this ILP without knowledge of a good solution up front. As a result, we are able to solve almost all our test instances to optimality.

2.4 Solving the ILP using an Iterative Method

In this section we describe our second method for solving the ILP to optimality. This method is constructed to mimic the Branch and Price idea, but without the complications that arise when columns are added continuously in the Branch and Cut tree.

The basic structure of the method is as follows. We initialize with a small number of columns, and start a Branch and Cut algorithm. Let the problem defined by the current set of columns be $\mathcal{P}$. To ensure that the solution found by the algorithm is optimal, when considering the complete set of columns, we must be sure that adding any additional column to $\mathcal{P}$ will not improve the objective value. To this goal, we do the following. In each node of the Branch and Cut tree, after solving the LP-relaxation of that node, we calculate reduced costs for all columns, and mark the ones with negative reduced cost that are not already in $\mathcal{P}$. When enough columns are marked, we break the Branch and Cut algorithm and add the marked columns, or a subset of these, to $\mathcal{P}$. With this increased set of columns we start a new Branch and Cut. After a number of iterations, the algorithm will either terminate with $\mathcal{P}$ solved to optimality and no marked columns, or terminate due to memory restrictions. In the former case, we have a proven optimal solution for the complete set of columns. In the latter case, as long as there are no marked columns, the global lower bound obtained so far with the procedure will be a global lower bound for the complete problem. Furthermore, the best know feasible solution in $\mathcal{P}$ is a feasible solution for the complete problem. These facts are true since when there are no mark columns this corresponds to partially solving a complete problem with a Branch and Cut algorithm.

To construct the initial ILP problem $\mathcal{P}$, we do as follows. Clearly the basis from the optimal LP-relaxation should be included to ensure that the global lower bound of the Branch and Cut tree is as high as possible. Furthermore, to ensure integer feasibility, we include the set of unit columns. If b is the number of columns in the basis, this gives us in total $2b$ columns. We want to add a small number of additional columns to improve the chance of obtaining a good integer solution in the first iteration. We choose these columns among the set of columns that were marked during the solution of the LP-relaxation, since these are the columns that have appeared to be attractive at some point. Let $\mathcal{M}$ be the set of marked columns not in $\mathcal{P}$. For all columns, j in $\mathcal{M}$, we have $\overline{c}_j^{\rho} \geq 0$ with respect to the optimal LP-relaxation basis. We initialize a threshold value and scan through the columns in $\mathcal{M}$. During this process we add column j to $\mathcal{P}$ if $\overline{c}_j^{\rho}$ is below the threshold value. If less than b columns are added in this process, we increase the threshold and scan the columns again. This gives us a problem $\mathcal{P}$ consisting of attractive and useful columns.

For adding columns between two iterations, we specify two parameters as follows. Let w_b be the threshold value for the number of columns we want to mark before the Branch and Cut algorithm is broken, and let w_m be the number of columns that are actually marked. During the Branch and Cut algorithm we sum up the number of columns in $\mathcal{M}$ that we encounter with negative reduced cost with respect to the optimal LP-relaxation of each node. When at least w_b columns are marked during this process, we break the algorithm. If $w_m \geq 1.5w_b$, all marked columns are added to $\mathcal{P}$. Otherwise we add those with the smallest reduced cost, using a threshold method similar to the one used in the initialization, until at least $0.75w_b$ columns are added. If the problem with the current set, $\mathcal{P}$, is solved to optimality, we decrease w_b by a factor 10. Otherwise w_b is increased by a factor two. Though, if the current problem

is terminated due to memory restrictions, we set w_b to a small number to ease the search for a valid global lower bound.

3 Pref Follow-on Heuristic

Large instances and instances with wide time windows are very hard to solve to optimality. In this section, we turn our attention to sub optimal solution of the CARP-TW as we present a heuristic for the problem which is based on the optimal solution strategy just presented, but with a significantly smaller number of columns.

The idea is to mimic the "fixing follow-on" idea from Airline Scheduling, [1], by limiting the set of edges that may follow a given edge on a vehicle tour. This is done by first associating a list, $\mathcal{F}_e$, with every required edge, e, of the graph. This list contains every edge, e', that may be serviced immediately after e due to some criteria which will be defined subsequently. All feasible tours that respect these lists in addition to the time window and capacity restrictions are then constructed. This gives us a set of feasible vehicle tours. Among these tours an optimal subset is chosen to make up the final solution. This subset is chosen in such a way that every required edge is serviced by exactly one tour. The algorithm is outlined as follows, where the details will be given below.

1. Construct a list $\mathcal{F}_e$ for each required edge e.
2. Construct the set, $\mathcal{E}$, of feasible tours that respects the lists $\mathcal{F}_e$.
3. Choose a subset, $\mathcal{E}'$, of $\mathcal{E}$ to be a complete solution to the problem

The criteria for construction of the $\mathcal{F}_e$ lists are essential for the quality of the solution. First, the dead heading between any two edges being serviced after each other, should not be too long. We incorporate this distance issue into criteria 1 - 3 below. Secondly, the waiting time between the service of two edges should not be too long, since this could cause the vehicle to run out of time, and hence, extra vehicles being needed to cover the remaining required edges. In criteria 4 and 5 we incorporate both the distance issue and the time issue. The resulting criteria are as follows.

1. $e' \in \mathcal{F}_e$ **if** $\textbf{dist}(e \to e') \leq \frac{1}{2}\beta/(n_d\gamma\ )$
2. $e' \in \mathcal{F}_e$ **if** $\textbf{dist}(e \to e') \leq \frac{1}{2}\beta/(n_d n)$
3. $e' \in \mathcal{F}_e$ **if** $\textbf{dist}(e \to e') \leq \frac{1}{2}\beta/n^2$
4. $e' \in \mathcal{F}_e$ **if** $\textbf{dist}(e \to e') \leq 0.6\beta/(n_d\gamma\ )$ **and** $\textbf{wait}(e \to e') \leq \textbf{AW}$
5. $e' \in \mathcal{F}_e$ **if** $\textbf{dist}(e \to e') \leq 0.6\beta/(n_d n)$ **and** $\textbf{wait}(e \to e') \leq \textbf{AW}$

where $\textbf{dist}(e \to e')$ is the shortest distance between any pair of end nodes of e and e', $\beta = \sum_{i=1}^{n}\sum_{j=1}^{n} SPL(i,j)$, γ is the length of the longest shortest path between any pair of nodes in the graph, i.e. $\max_{i,j\in N} SPL(i,j)$, and n_d is the number of required edges in the graph. Furthermore, $\textbf{wait}(e \to e')$ is the shortest waiting time that must occur if e' is serviced immediately after e, and AW is the average value of $\textbf{wait}(e \to e')$, where the average is for pairs of e and e' that are feasible with respect to time windows and vehicle capacity.

For the construction of the set, $\mathcal{E}$, of all feasible tours, we follow the same depth first method as explained in Section 2.1, with the exception that now the $\mathcal{F}_e$ lists are taken into account. Doing this, we ensure that all tours in $\mathcal{E}$ respect the lists $\mathcal{F}_e$ in addition to being feasible with respect to time windows and vehicle capacity. To

determine the set $\mathcal{E}'$ to be the final set of columns, we use the Two-Phase Method described in Section 2.3. As the number of columns in $\mathcal{E}$ is significantly smaller than for the problems considered in Section 2.3, $\mathcal{E}'$ can be determined in time competitive to that of other heuristics for the CARP-TW.

4 Computational Results

We have compared our methods on a set of 20 instances with varying time window sizes. All instances are based on the so-called Eglese instances for the classical CARP, and contain from 51 to 190 required edges. They are available at [25]. Results obtained with a preliminary version of the 2-Phase algorithm on the set of smaller A, B, and C instances, also available in [25], are given in [26]. All results are obtained using C++ on a Pentium M, 1500 MHz processor and 480 MB RAM. Details about the test instances can be found in Table 1, where the first three columns give the number of nodes, number of edges, and number of required edges, and the following five columns give detailed information about the instance (average demand of the required edges, vehicle capacity, Φ, Ψ, and average length of the time windows). The remaining three columns give information about the corresponding Set Partitioning Problem. This information consists of number of rows and legal columns in the problem along with the density of the problem, i.e. the percentage of non-zeros in the constraint matrix.

Table 1: Details about the instances.

	n	m	m_d	ad	W	Φ Ψ	tw	rows	cols	density
TW-e1	77	98	51	28.8	305	1 2	17.2	51	2428587	7.13
TW-e2	77	98	51	28.8	220	1 2	17.2	51	2071683	6.95
TW-e3	77	98	51	28.8	160	1 2	17.2	51	1141252	6.50
TW-e4	77	98	72	26.1	280	1 2	5.3	72	1066627	6.47
TW-e5	77	98	72	26.1	200	1 2	5.3	72	1060031	6.47
TW-e6	77	98	72	26.1	140	1 2	5.3	72	908131	6.39
TW-e7	77	98	87	25.1	280	1 2	16.0	87	135173	5.37
TW-e8	77	98	87	25.1	190	1 2	16.0	87	135168	5.37
TW-e9	77	98	87	25.1	135	1 2	16.0	87	133808	5.37
TW-e10	77	98	98	25.0	280	1 2	5.1	98	63659	4.66
TW-e11	77	98	98	25.0	180	1 2	5.1	98	63607	4.66
TW-e12	77	98	98	25.0	130	1 2	5.1	98	61994	4.65
TW-s1	140	190	75	18.6	210	1 3	16.9	75	8883219	7.13
TW-s2	140	190	75	18.6	150	1 3	16.9	75	8258973	7.03
TW-s3	140	190	147	21.6	160	1 3	4.7	147	6221028	6.09
TW-s4	140	190	159	21.3	160	1 3	15.3	159	186828	4.73
TW-s5	140	190	159	21.3	120	1 3	15.3	159	185701	4.73
TW-s6	140	190	190	22.0	230	1 3	4.6	190	124208	4.07
TW-s7	140	190	190	22.0	160	1 3	4.6	190	122693	4.06
TW-s8	140	190	190	22.0	120	1 3	4.6	190	116243	4.05

Algorithms for solving problems to optimality can be compared on two dimensions: running time and ability to solve hard instances. The running time comparison is based on the total time the algorithms take to solve the problem instances to optimality. Comparing the ability to solve problems is based on how well the algorithms can solve the problems to optimality if given infinite time. Since this is impractical, a time limit (e.g. 6 hours) are often given in practice, and the best integer solutions found by the algorithms within that time are compared.

When we consider the Set Partitioning version of the problems, 15 of our test instances could be solved with all three algorithms. For these instances, the algorithms are compared by running time. For the remaining 5 instances, at least one of the algorithms could not solve the problem to optimality. For these instances we compare the algorithms by ability to solve hard problems.

Table 2: Results for instances solved to optimality by all three methods.

	Value		Computation time			
	Integer	LP relax	columns	B & B	2-Phase	Iterative
TW-e3	6732	6667	37	6626	10034	**66**
TW-e4	9984	9961	35	3436	2750	**58**
TW-e5	9984	9961	35	6047	2139	**97**
TW-e6	10125	10101	28	1430	1483	**58**
TW-e7	13736	13723	2	**21**	414	37
TW-e8	13736	13723	3	19	12	**5**
TW-e9	13740	13736	3	20	12	**6**
TW-e10	15024	15007	1	**10**	48	27
TW-e11	15024	15007	1	**11**	33	28
TW-e12	15180	15180	1	**6**	32	14
TW-s4	25593	25368	3	42	58	**8**
TW-s5	25628	25419	3	**42**	867	115
TW-s6	33671	33671	1	**10**	15	38
TW-s7	33671	33671	2	**14**	422	47
TW-s8	33941	33941	1	15	8	**7**

Table 2 reports the results obtained for the 15 instances, where the Set Partitioning version could be solved to optimality by the standard CPLEX method. The first two columns give the value of the optimal integer solution and the value of the LP relaxation, respectively. The third column gives the time for generating the columns. The last three columns report the best computation time obtained with each of the three methods. These times are given in seconds and include the time for generating the columns. The best computation time for each instance is marked in bold. The general conclusions drawn from these results are that the standard method and the iterative method share the leading position with no clear picture of which one is the best. When we consider the remaining 5 instances we see the reason why the iterative method, in general, is a better choice. The second conclusion is that the 2-Phase method is not a candidate to be the best method, but for instances where the iterative method is the best, the 2-Phase method often comes in second, and hence is better than the standard method.

The results reported in the column labeled "B & B" are the results obtained by giving all columns to CPLEX 8.1 integer solver directly, and using the standard settings to solve the Set Partitioning Problem. The only type of cuts that is identified for this problem is the clique cuts. Hence the four different cut settings for this type of cuts are tested, and the running time reported is the minimum of the four. It was observed that here the fastest strategy is to generate no cuts at all. We believe that this is due to the fact that the number of rows is very small compared to the number of columns, and hence adding even a small number of cuts will highly increase the complexity of each simplex iteration within the B & B tree.

The results reported in the columns labeled "2-Phase" and "Iterative" are the best results obtained with different versions of the two-phase method presented in Section 2.3, and the iterative method presented in Section 2.4, respectively. For each of these methods we again tried different cut strategies. The conclusion drawn from

these tests are not as clear as those drawn from the B & B method. For the two-phase method, the computation time does not vary much with the number of cuts generated, though there is a tendency of slightly shorter computation times, when cuts are generated in the second phase. For the iterative method the tendency is that the further along in the solution process, the more useful the cutting, and hence, the best strategy is to increase the use of cuts along with the process.

For both methods, we have tested the use of several different branching strategies. Besides the CPLEX standard strategy, we have implemented a strategy, where branching is performed on sos-sets whenever possible. Finally, we have found inspiration for a branching strategy in the airline scheduling literature [1, 9, 11] and have implemented a branching scheme that mimics branching on follow-on. Here, as well, the branch is based directly in the original variables. Let e and e' be the row index for two required edges. We know that for each of these, exactly one column, j, that has $\alpha_{ej} = 1$ ($\alpha_{e'j} = 1$) must be used. We branch, so that columns in $\{j \mid \alpha_{ej} = 1$ and $\alpha_{e'j} = 1\}$ are in one branch, and columns in $\{j \mid (\alpha_{ej} = 1$ and $\alpha_{e'j} = 0)$ or $(\alpha_{ej} = 0$ and $\alpha_{e'j} = 1)\}$ are in the other branch, i.e. either the two edges are serviced by the same vehicle or they are serviced by different vehicles. Our results show that this strategy outperforms the other branching strategies tested.

For the two-phase method, we tested various strategies for choosing the set of columns to be contained in the first IP. The strategy that performed best was setting the desired number of columns to be between 100 and 2000 depending on the number of possible columns, though at least 10 times as many as the number of rows. If more than the desired number of columns were marked during the solution of the LP-relaxation, we randomly choose between the marked columns to obtain the desired number.

For the iterative method, we tested various combinations of choosing the number of marked columns before the B & B procedure is broken, and choosing the number of marked columns to add to the problem when the procedure is broken. With respect to choosing the number of columns to mark before breaking, the strategy that gave the best results was to start with a small number of columns, e.g. twice as many as the number of rows in the problem. If the sub problem is solved to optimality in the current iteration, the number of desired columns is decreased by a factor 10 for the next iteration, if not, the number of desired columns is doubled. Though, the first very small problem is not broken until an integer solution is found. When columns are added to the problem, we found the best strategy to be the following: add all marked columns if there are no more than 1.5 times the number of desired columns, and otherwise add between 2/3 and 4/3 times the desired number of columns while adding columns with largest absolute value of the reduced cost first.

Table 3: Results for instances that could not be solved by the standard B & B.

		Best integer solution obtained				Time	
	LP rel.	B & B	2-Phase	Iterative	Columns	2-Phase	Iterative
TW-e1	6330	O.M.	6915	***6423**	115	1659	356
TW-e2	6330	O.M.	***6424**	***6424**	85	913	201
TW-s1	8669	O.M.	***8738**	***8738**	554	13491	41257
TW-s2	8820	O.M.	**8946**	9030	391	12889	time
TW-s3	19565	O.M.	***19587**	***19587**	170	10375	20243

In Table 3 we consider the instances that could not be solved by giving all columns to the standard CPLEX 8.1 MIP solver due to memory restrictions. For such instances alternative methods are needed. In the table we give the best integer values obtained by the two-phase method and the iterative method, respectively. The best

results are shown in bold and an asterisk indicates that the integer solution found is known to be optimal. The O.M. in the B & B column indicates the out of memory situation. For all five instances, the two-phase method ran out of memory in the initial part of the second stage. The fifth column gives the column generation time and the last two columns give the computation times used to obtain these results. The iterative method reached its time limit of 12 hours for one instance. For the remaining instances, this method solved the problem to optimality within the time limit. For the two-phase method, the given time is the time before the out of memory situation. This method obtained the optimal solution for three instances, but did not prove the optimality. The first column of the table gives the value of the LP-relaxation of the Set Partitioning version of the problems. All settings for the two methods are as explained above.

Table 4: Computational Results for the Preferable Follow-on Heuristic

	Best	Solution value obtained					Percentage above best known				
	known	p1	p2	p3	p4	p5	p1	p2	p3	p4	p5
e1	*6423	8789	**6447**	**6447**	8293	6448	36,84	0,37	0,37	29,11	0,39
e2	*6424	8789	6475	6517	8293	**6448**	36,82	0,79	1,45	29,09	0,37
e3	*6732	8990	***6732**	6856	8521	6808	33,54	0,00	1,84	26,57	1,13
e4	*9984	14111	10546	10546	13755	***9984**	41,34	5,63	5,63	37,77	0,00
e5	*9984	14111	10546	10546	13755	***9984**	41,34	5,63	5,63	37,77	0,00
e6	*10125	14377	10746	10746	13999	**10360**	42,00	6,13	6,13	38,26	2,32
e7	*13736	17266	14096	**14032**	17202	14060	25,70	2,62	2,15	25,23	2,36
e8	*13736	17266	14096	**14032**	17202	14060	25,70	2,62	2,15	25,23	2,36
e9	*13740	17266	14096	**14032**	17202	14060	25,66	2,59	2,13	25,20	2,33
e10	*15024	18915	15782	**15534**	18630	18630	25,90	5,05	3,39	24,00	24,00
e11	*15024	18915	15782	**15534**	18630	16090	25,90	5,05	3,39	24,00	7,10
e12	*15180	18895	15949	**15699**	18666	16267	24,47	5,07	3,42	22,96	7,16
s1	*8738	10397	**8738**	8951	9818	8745	18,99	0,00	2,44	12,36	0,08
s2	8946	10397	**8897**	9184	10228	9036	16,22	-0,55	2,66	14,33	1,01
s3	*19587	25534	19891	**19587**	25155	22617	30,36	1,55	0,00	28,43	15,47
s4	*25593	31162	**25692**	**25692**	29821	25708	21,76	0,39	0,39	16,52	0,45
s5	*25628	31222	25725	**25723**	29843	25741	21,83	0,38	0,37	16,45	0,44
s6	*33671	38883	34163	**34135**	38310	34432	15,48	1,46	1,38	13,78	2,26
s7	*33671	38883	34163	**34135**	38310	34432	15,48	1,46	1,38	13,78	2,26
s8	*33941	39132	34448	**34365**	38565	34662	15,29	1,49	1,25	13,62	2,12
av							27.03	2.39	2.38	23.72	3.68

In section 3, we presented the Preferable Follow-on heuristic for solving the CARP-TW, which is based on our two-phase method for solving the Set Partitioning Problem to optimality. In Table 4 we give our computational results obtained with this heuristic.

The first column of the table gives the value of the best solution known so far. This solution is the best one obtained with the exact methods presented in this paper. The solution is marked with an asterisk when it is known to be optimal, which is the case for all but one instance.

The next part of the table gives the results obtained with the five versions of the heuristic. To obtain these results we use the two-phase method with the parameter setting that according to the experience worked best in general. Here the best result is shown in bold, and an asterisk indicates that the heuristic reached a proven optimal solution. Note that for instance s2, the result obtained with version 2 of the heuristic is better than the best solution known so far. This is the instance, where we did not reach the optimal solution due to the time limit.

The second part of Table 4 gives the results as percentage above the best known solution, and Table 5 gives the computation time in seconds used to obtain the results. Here, T-21600 indicates that the time limit of 6 hours has been reached.

Table 5: Computation time for the Preferable Follow-on Heuristic

	pref1	pref2	pref3	pref4	pref5
e1	0	8685	120	10	T-21600
e2	1	8602	111	2	T-21600
e3	1	238	217	9	T-21600
e4	2	102	64	2	12762
e5	1	79	75	2	339
e6	0	4745	720	10	19855
e7	3	25	56	9	167
e8	0	12	17	10	117
e9	1	25	47	9	55
e10	1	4	40	10	6
e11	3	18	150	10	319
e12	1	14	15	9	39
s1	1	3046	24	5	9450
s2	3	572	105	73	7702
s3	1	5115	6231	10	361
s4	1	76	380	10	25
s5	2	19	82	10	97
s6	1	8	25	9	20
s7	0	8	16	9	2
s8	1	6	18	9	5

5 Conclusion

We have presented two column generation methods for solving the Capacitated Arc Routing Problem with Time Windows and a heuristic to obtain near optimal solutions to the problem. Both methods are significantly better than the standard method of solving the complete Set Partitioning Problem as both methods can solve problems that the standard cannot. The iterative method is superior to the two-phase method both regarding computation time and ability to solve hard instances. Five versions of our Preferable Follow-on Heuristics were tested. The computational results regarding best integer solution found favor versions 2, 3, and 5. When taking computation time into account, version 2 and in particular 3 are superior to the other versions.

References

[1] Ranga Anbil, Ellis L. Johnson, and Rajan Tanga. A Global Approach to Crew-Pairing Optimization. *IBM Systems Journal*, 31(1), 1992.

[2] Arjang A. Assad, Bruce L. Golden, and Wen-Lea Pearn. Transforming Arc Routing into Node Routing Problems. *Computers and Operations Research*, 14(4):285–288, 1987.

[3] Roberto Baldacci and Vittorio Maniezzo. Exact Methods Based on Node Routing Formulations for Undirected Arc Routing Problems. *Networks*, 47(1):52–60, 2006.

[4] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, and Martin W.P. Savelsberg. Branch-and-Price: Column Generation for Solving Huge Integer programs. *Operations Research*, 46(3):316–329, 1998.

[5] José M. Belenguer and Enrique Benavent. The Capacitated Arc Routing Problem: Valid Inequalities and Facets. *Computational Optimization and Applications*, 10:165–187, 1998.

[6] José M. Belenguer and Enrique Benavent. A Cutting Plane Algorithm for the Capacitated Arc Routing Problem. *Computers and Operations Research*, 30(5):705–728, 2003.

[7] José M. Belenguer, Enrique Benavent, and David Gómez-Cabrero. Cutting Plane and Column Generation for the Capacitated Arc Routing Problem. *ORP3 Meeting, Valencia*, 2005.

[8] Lawrence D. Bodin and Samuel J. Kursh. A Computer-Assisted System for the Routing and Scheduling of Street Sweepers. *Operations Research*, 26(4):525–537, 1987.

[9] H.D. Chu, E. Gelman, and Ellis L. Johnson. Solving large scale crew scheduling problems. *European Journal of Operational Research*, 97:260–268, 1997.

[10] Lloyd Clarke, Ellis Johnson, George Nemhauser, and Zhongxi Zhu. The Aircraft Rotation Problem. *Anals of Operations Research*, 69:33–46, 1997.

[11] L.W. Clarke, C.A. Hane, Ellis L. Johnson, and George L. Nemhauser. Maintenance and Crew Considerations in Fleet Assignment. *Transportation Science*, 30(3):249–260, 1996.

[12] George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8:101–111, 1960.

[13] George B. Dantzig and Philip Wolfe. The Decomposition Algorith for Linear Programs. *Econometroca*, 29(4):767–778, 1961.

[14] Marcus Poggi de Aragão, Humberto Longo, and Eduardo Uchoa. Solving Capacitated Arc Routing Problem using a Transformation to the CVRP. *Computers and Operations Research*, 33(6):1823–1837, 2006.

[15] Richard W. Eglese. Routing Winter Gritting Vehicle. *Discrete Applied Mathematics*, 48:231–244, 1994.

[16] Eric Gelman, Diego Klabjan, Ellis L. Johnson, George L. Nemhauser, and Srini Ramaswamy. Airline Crew Acheduling with Time Windows and Plane Count Constraints. *Transportation Science*, 36(3):337–348, 2002.

[17] Z. Gu, George L. Nemhauser, and M. Savelsberg. Sequence Independent Lifting in Mixed Integer Programing. *Journal of Combinatorial Optimization*, 4:109–129, 2000.

[18] Ryuichi Hirabayashi, Naonori Nishida, and Yasufumi Saruwatari. Tour Construction Algorithm for the Capacitated Arc Routing Problem. *Asia-Pacific Journal of Operational Research*, 9:155–175, 1992.

[19] Jing Hu and Ellis L. Johnson. Computational results with a primal-dual subproblem simplex method. *Operations Research Letters*, 25:149–157, 1999.

[20] Ellis L. Johnson, D. Klabjan, and George L. Nemhauser. A Parallel Primal-Dual Simplex Algorithm. *Technical Report, Georgia Institute of Technology*, 1999.

[21] Nacima Labadi, Christian Prins, and Mohamed Reghioui. GRASP with Path Relinking for the Capacitated Arc Routing Problem with Time Windows. *Lecture Notes in Computer Science*, 4448:722–731, 2007.

[22] Adam N. Letchford. *Polyhedral Results for Some Constrained Arc-Routing Problems.* PhD thesis, Lancaster University, 1996.

[23] Adam N. Letchford and Amar Oukil. Exploiting Sparsity in Pricing Routines for the Capacitated Arc Routing Problem. *Working paper, Department of Management Science, Lancaster University*, 2006.

[24] Paul Abraham Mullaseril. *Capacitated Rural Postman Problem with Time Windows and Split Delivery.* PhD thesis, MIS Department, University of Arizona, Tucson, Arizona 85721, 1997.

[25] Sanne Wøhlk. http://www.wohlk-son.dk/sanne/research_carptw.html.

[26] Sanne Wøhlk. *Contributions to Arc Routing. Ph.D. diss.* PhD thesis, University of Southern Denmark, 2005.

[27] G. Yu, editor. *Operations Research in the Airline Industry.* Kluwer Academic Publishers, 1998.

Working Papers from CORAL – Centre for Operations Research Applications in Logistics

L-2008-09 Ellis L. Johnson & Sanne Wøhlk: Solving the Capacitated Arc Routing Problem with Time Windows using Column Generation.

L-2008-08 Søren Bloch & Christian H. Christiansen: Simultaneously Optimizing Storage Location Assignment at Forward Area and Reserve Area – a Decomposition Based Heuristic.

L-2008-07 Bisheng Du & Christian Larsen: Base stock policies with degraded service to larger orders.

L-2008-06 Karina H. Kjeldsen: Classification of routing and scheduling problems in liner shipping.

L-2008-05 Daniele Pretolani, Lars Relund Nielsen, Kim Allan Andersen & Matthias Ehrgott: Time-adaptive versus history-adaptive strategies for multicriterion routing in stochastic time-dependent networks.

L-2008-04 Christian Larsen: Comments to Özkaya et al. (2006).

L-2008-03 Christian Larsen: Derivation of confidence intervals of service measures in a base-stock inventory control system with low-frequent demand.

L-2008-02 Jens Lysgaard: The Pyramidal Capacitated Vehicle Routing Problem.

L-2008-01 Jens Lysgaard & Janni Løber: Scheduling participants of Assessment Centres.

L-2007-03 Christian Larsen: Note: Comments on the paper by Rosling (2002).

L-2007-02 Christian Larsen, Claus Hoe Seiding, Christian Teller & Anders Thorstenson: An inventory control project in a major Danish company using compound renewal demand models.

L-2007-01 Christian Larsen: The *Q(s,S)* control policy for the joint replenishment problem extended to the case of correlation among item-demands.

L-2006-11 Daniele Pretolani, Lars Relund Nielsen & Kim Allan Andersen: A note on “Multicriteria adaptive paths in stochastic, time-varying networks”.

L-2006-10 Lars Relund Nielsen, Kim Allan Andersen & Daniele Pretolani: Bicriterion a priori route choice in stochastic time-dependent networks.

L-2006-09 Christian Larsen & Gudrun P. Kiesmüller: Developing a closed-form cost expression for an *(R,s,nQ)* policy where the demand process is compound generalized Erlang.

L-2006-08 Eduardo Uchoa, Ricardo Fukasawa, Jens Lysgaard, Artur Pessoa, Marcus Poggi de Aragão, Diogo Andrade: Robust Branch-Cut-and-Price for the Capacitated Minimum Spanning Tree Problem over a Large Extended Formulation.

L-2006-07 Geir Brønmo, Bjørn Nygreen & Jens Lysgaard: Column generation approaches to ship scheduling with flexible cargo sizes.

L-2006-06 Adam N. Letchford, Jens Lysgaard & Richard W. Eglese: A Branch-and-Cut Algorithm for the Capacitated Open Vehicle Routing Problem.

L-2006-05 Ole Mortensen & Olga W. Lemoine: Business integration between manufacturing and transport-logistics firms.

L-2006-04 Christian H. Christiansen & Jens Lysgaard: A column generation approach to the capacitated vehicle routing problem with stochastic demands.

L-2006-03 Christian Larsen: Computation of order and volume fill rates for a base stock inventory control system with heterogeneous demand to investigate which customer class gets the best service.

L-2006-02 Søren Glud Johansen & Anders Thorstenson: Note: Optimal base-stock policy for the inventory system with periodic review, backorders and sequential lead times.

L-2006-01 Christian Larsen & Anders Thorstenson: A comparison between the order and the volume fill rates for a base-stock inventory control system under a compound renewal demand process.

L-2005-02 Michael M. Sørensen: Polyhedral computations for the simple graph partitioning problem.

L-2005-01 Ole Mortensen: Transportkoncepter og IT-støtte: et undersøgelsesoplæg og nogle foreløbige resultater.

L-2004-05 Lars Relund Nielsen, Daniele Pretolani & Kim Allan Andersen: *K* shortest paths in stochastic time-dependent networks.

L-2004-04 Lars Relund Nielsen, Daniele Pretolani & Kim Allan Andersen: Finding the *K* shortest hyperpaths using reoptimization.

L-2004-03 Søren Glud Johansen & Anders Thorstenson: The (*r*,*q*) policy for the lost-sales inventory system when more than one order may be outstanding.

L-2004-02 Erland Hejn Nielsen: Streams of events and performance of queuing systems: The basic anatomy of arrival/departure processes, when focus is set on autocorrelation.

L-2004-01 Jens Lysgaard: Reachability cuts for the vehicle routing problem with time windows.

ISBN 9788778823618

Department of Business Studies

Aarhus School of Business
University of Aarhus
Fuglesangs Allé 4
DK-8210 Aarhus V - Denmark

Tel. +45 89 48 66 88
Fax +45 86 15 01 88

www.asb.dk