The Manpower Allocation Problem with Time Windows and Job-Teaming Constraints: A Branch-and-Price Approach

Anders Dohn*, Esben Kolind, and Jens Clausen
Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark

March 12, 2007

In this paper, we consider the Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams (m-MAPTWTC). Given a set of teams and a set of tasks, the problem is to assign to each team a sequential order of tasks to maximize the total number of assigned tasks. Both teams and tasks may be restricted by time windows outside which operation is not possible. Some tasks require cooperation between teams, and all teams cooperating must initiate execution simultaneously. We present an IP-model for the problem, which is decomposed using Dantzig-Wolfe decomposition. The problem is solved by column generation in a Branch-and-Price framework. Simultaneous execution of tasks is enforced by the branching scheme. To test the efficiency of the proposed algorithm, 12 realistic test instances are introduced. The algorithm is able to find the optimal solution in 11 of the test instances. The main contribution of this article is the addition of synchronization between teams in an exact optimization context.

1 Introduction and Problem Description

The Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams (m-MAPTWTC) is the problem of assigning m teams to a number of tasks, where both teams and tasks may be restricted by time windows outside which operation is not possible. Tasks may require several individual teams to cooperate. Due to the limited number of teams, some tasks may have to be left unassigned. The objective is to maximize the number of assigned tasks.

The problem arises in various contexts where cooperation between teams/workers, possibly with different skills, is required to solve tasks. An example is the home care sector, where the personnel travel between the homes of the patients who may demand collaborative work (e.g. for lifting). The problem also occurs in hospitals where a number of doctors and nurses are needed for surgery and the composition of staff may vary for different tasks. Another example is in the allocation of technicians to service jobs, where a combination of technicians with individual skills is needed to solve each task.

*Corresponding author. E-mail address: adh@imm.dtu.dk

This study focuses on the scheduling of ground handling tasks in some of Europe's major airports. Between arrival and the subsequent departure of an aircraft, numerous jobs including baggage handling and cleaning must be performed. Typically, specialized handling companies take on the jobs and assign crews of workers with different skills. Any daily work plan must comply with the time windows of tasks, the working hours of the staff, the skill requirements of tasks, and union regulations. It may be necessary to have several teams cooperating on one task in order to complete it within the time window. The workload has to be divided equally among the cooperating teams. Furthermore, all teams involved must initiate work on the task simultaneously (synchronized cooperation), as only one of the team leaders is appointed as responsible supervisor. In the remainder of this paper, a team is a fixed group of workers, whereas when referring to job-teaming or cooperation, we refer to a temporary constellation of teams joined together for a specific task. In the airport setting, all tasks require exactly one skill each.

MAPTWTC has previously been treated by Lim et al. [28] and Li et al. [26] in a metaheuristic approach. They study an example originating from the Port of Singapore, where the main objective is to minimize the number of workers required to carry out all tasks, rather than carrying out the maximum number of tasks with a given workforce. Both papers describe secondary objectives as well.

Our problem is closely related to the Vehicle Routing Problem with Time Windows (VRPTW) which has been studied extensively in the literature.

The Vehicle Routing Problem with Split Deliveries (VRPSD) allows a customer to be visited by several vehicles, each fulfilling some of the demand. The problem was introduced by Dror and Trudeau [10]. See [25] for an overview of the literature. Frizzell and Giffin [13] were the first to include the time window extension in the split delivery problem (VRPTWSD). They solve the problem heuristically. A tabu search for VRPTWSD is developed by Ho and Haugland [16].

Lau et al. [24] formulate the vehicle routing problem with time windows and a limited number of vehicles (m-VRPTW) and solve it using a tabu search approach. See [29] and [27] for other heuristic approaches to the same problem.

In order to find literature on exact solution to VRPTW, we have to revert to a formulation without split deliveries and without limitations on the number of vehicles. The most promising recent results for exact solution to VRPTW problems use column generation. Column generation for VRPTW was initiated by Desrochers et al. [8]. They solve the pricing problem as a *Shortest Path Problem with Time Windows (SPPTW)*. Their approach proved very successful and was further applied and developed by Kohl [21], Kohl et al. [22], Larsen [23], Cook and Rich [5], Kallehauge et al. [19], Righini and Salani [30], and Irnich and Villeneuve [17].

Recently, Feillet et al. [12] suggested solving the pricing problem as an *Elementary Shortest Path Problem with Time Windows (ESPPTW)* building on the ideas of Beasley and Christofides [2]. Chabrier [3], Danna and Pape [6], and Jepsen et al. [18] have extended the ideas and achieved very promising results.

The remainder of this paper is structured as follows. In Section 2, we present an IP formulation of m-MAPTWTC. In Section 3, the formulation is decomposed into a master problem and a pricing problem using Dantzig-Wolfe decomposition. This decomposition allows us to solve the problem using column generation in a Branch-and-Price framework. In Section 4, the necessary branching rules are described. This includes branching to enforce integrality as well as synchronized cooperation on tasks. The computational results on a number of real-life problems are presented in Section 5. Finally, in Section 6 we conclude on our work and discuss possible areas for future research.

2 Problem Definitions and Formulation

2.1 IP Formulation of m-MAPTWTC

Consider a set C of n tasks and a workforce of inhomogeneous teams V. All shifts begin at a service center, referred to as location 0. The set of tasks together with the service center is denoted N. For each task $i \in C$ a time window is defined as $[a_i, b_i]$ where a_i and b_i are the earliest and the latest starting times for task i, respectively. r_i is the number of teams required to carry out task i (Task i is divided into r_i *split tasks*). Each team $k \in V$ has a time window $[e_k, f_k]$, where the team starts at the service center at time e_k and must return no later than f_k. Between each pair of tasks (i, j), we associate a time t_{ij} which contains the travel time from i to j and the service time at task i. Further, g_{ik} is a binary parameter defining whether team k has the required qualifications for task i ($g_{ik} = 1$) or not ($g_{ik} = 0$).

We assume that a_i, b_i, e_k, f_k are non-negative integers and that each t_{ij} is a positive integer. We also assume that the triangular inequality is satisfied for t_{ij}.

To solve the problem, two sets of decision variables have to be defined:

x_{ijk} is binary with $x_{ijk} = \begin{cases} 1, & \text{if team } k \text{ goes directly from task } i \text{ to task } j. \\ 0, & \text{otherwise} \end{cases}$

s_i is an integer variable and defines the starting time of task i.

m-MAPTWTC can be formulated mathematically as:

$$\max \sum_{k \in V} \sum_{i \in C} \sum_{j \in N} x_{ijk} \tag{1}$$

$$\sum_{k \in V} \sum_{j \in N} x_{ijk} \leq r_i \qquad \forall i \in C \tag{2}$$

$$x_{ijk} \leq g_{ik} \qquad \forall i \in C, \forall j \in C, \forall k \in V \tag{3}$$

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall k \in V \tag{4}$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \qquad \forall h \in N, \forall k \in V \tag{5}$$

$$e_k + t_{0j} - M(1 - x_{0jk}) \leq s_j \qquad \forall j \in N, \forall k \in V \tag{6}$$

$$s_i + t_{i0} - M(1 - x_{i0k}) \leq f_k \qquad \forall i \in N, \forall k \in V \tag{7}$$

$$s_i + t_{ij} - M(1 - x_{ijk}) \leq s_j \qquad \forall i \in C, \forall j \in C, \forall k \in V \tag{8}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in N, \forall k \in V \tag{9}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in N, \forall j \in N, \forall k \in V \tag{10}$$

$$s_{ik} \in \mathbb{Z}^+ \cup \{0\} \qquad \forall i \in N, \forall k \in V \tag{11}$$

The objective (1) is to maximize the number of assigned tasks. A task is counted multiple times if split between teams ($r_i > 1$). The constraints (2) guarantee that each task is assigned the right number of teams or possibly less, if some of its split tasks are left unassigned. Only teams with the required skill can be assigned to a specific task (3). Furthermore, we have to ensure that all shifts start in the service center (4). Constraints (5) ensure that no shifts are segmented. Any task visited by a team must be left again. The next four constraints deal

with the time windows. First, we ensure that a team can only be assigned to a task during their working hours (6)-(7). Next, we check if the time needed for traveling between tasks is available (8). If a customer i is not visited, the large scalar M makes the corresponding constraints non-binding. Constraints (9) enforce the task time windows. Finally, constraints (10)-(11) are the integrality constraints. The introduction of a service start time removes the need for sub-tour elimination constraints, since each customer can only be serviced once during the scheduling horizon because t_{ij} is positive.

2.2 Relations to Vehicle Routing

As mentioned earlier, m-MAPTWTC is closely related to VRPTW. Consider the teams as vehicles driving from one customer to another as they in m-MAPTWTC move from one task to another. The service that the teams deliver is an amount of their time, unlike the vehicles that deliver goods which have taken up a part of the total volume. Hence, in that sense m-MAPTWTC is uncapacitated. Except for the binding between teams inflicted by the possibility of cooperation on tasks, the problem is similar to the Uncapacitated Vehicle Routing Problem with Time Windows and a limited number of vehicles (m-VRPTW).

Column generation has proven a successful technique for exact solution of VRPTW and as m-MAPTWTC is also NP-hard (see [26]) the solution procedure in this article is built on the principles of column generation in a Branch-and-Price framework.

3 Decomposition

We present the Dantzig-Wolfe decomposition [7] of m-MAPTWTC. First, we introduce the notion of a *path*. A feasible path is defined as a shift starting and ending at the service center, obeying time windows and skill requirements, but disregarding the constraints dealing with interaction between shifts. By this definition the feasibility of a path can be determined without further knowledge about other paths. We define $\mathcal{P}_k$ as the set of all feasible paths for team $k \in V$. Let the set T_i be the set of all possible start times for task i. Each path is defined by the tasks it visits and the time of initiation of each task. Let $\hat{a}_{ik}^{pt} = 1$ if task i is initiated at time t on path p for team k and $\hat{a}_{ik}^{pt} = 0$ otherwise.

3.1 Master Problem

In the *integer master problem* we solve the problem of optimally choosing one feasible path for each team, maximizing the total number of assigned tasks. In the original formulation, the equations (3)-(9) are used to ensure feasibility of paths. In the master problem, the set $\mathcal{P}_k$ is used to guarantee this feasibility. The use of only one s_i for each task had the effect that cooperating teams would initiate work simultaneously. In the master problem this has to be enforced by a new binary decision variable γ_i^t.

Now, the integer programming master problem is formulated as below, where λ_k^p are binary variables, which for each vehicle k are used to select a path p from $\mathcal{P}_k$. γ_i^t is a binary variable deciding if task i is initiated at time t. Any feasible solution to the master problem is a feasible solution to the original formulation.

$$\max \sum_{k\in V}\sum_{i\in N}\sum_{p\in\mathcal{P}_k}\sum_{t\in T_i}\hat{a}_{ik}^{pt}\lambda_k^p \tag{12}$$

$$\sum_{k\in V}\sum_{p\in\mathcal{P}_k}\hat{a}_{ik}^{pt}\lambda_k^p \le r_i\gamma_i^t \qquad \forall i\in C, \forall t\in T_i \tag{13}$$

$$\sum_{t\in T_i}\gamma_i^t = 1 \qquad \forall i\in C \tag{14}$$

$$\sum_{p\in\mathcal{P}_k}\lambda_k^p = 1 \qquad \forall k\in V \tag{15}$$

$$\lambda_k^p \in \{0,1\} \qquad \forall k\in V, \forall p\in\mathcal{P}_k \tag{16}$$

$$\gamma_i^t \in \{0,1\} \qquad \forall i\in C, \forall t\in T_i \tag{17}$$

The objective still is to maximize the number of assigned tasks (12). (13) has two effects. For each team it ensures that a path can only be selected if all tasks in the path comply with their respective time of initiation. Further, it ensures that each task is not assigned more teams than requested. In (14) we force all tasks to have only one time of initiation, and (15) guarantees that all teams have exactly one path assigned to them.

To apply column generation, the integrality constraints are relaxed to allow solution of the master problem by a standard linear solver. Unfortunately, the γ_i^t-variables loose all significance when LP-relaxed. Consider the LP-relaxed problem, i.e. (12)-(15) with the relaxed constraints $0 \le \lambda_k^p \le 1, \forall k\in V, \forall p\in\mathcal{P}_k$ and $0\le\gamma_i^t\le 1, \forall i\in C, \forall t\in T_i$. The LP-problem is a relaxation of the following problem:

$$\max \sum_{k\in V}\sum_{i\in N}\sum_{p\in\mathcal{P}_k}\sum_{t\in T_i}\hat{a}_{ik}^{pt}\lambda_k^p \tag{18}$$

$$\sum_{k\in V}\sum_{p\in\mathcal{P}_k}\sum_{\forall t\in T_i}\hat{a}_{ik}^{pt}\lambda_k^p \le r_i \qquad \forall i\in C \tag{19}$$

$$\sum_{p\in\mathcal{P}_k}\lambda_k^p = 1 \qquad \forall k\in V \tag{20}$$

$$0 \le \lambda_k^p \le 1 \qquad \forall k\in V, \forall p\in\mathcal{P}_k \tag{21}$$

Proof. According to Wolsey [32]: A problem (PR) $z^R = \max\{f(x) : x\in T\subseteq\mathbb{R}^n\}$ is a relaxation of (P) $z=\max\{c(x) : x\in X\subseteq\mathbb{R}^n\}$ if:

1. $X\subseteq T$
2. $f(x)\ge c(x), \qquad \forall x\in X$

Take any feasible solution λ' to (18)-(21). Set each γ'^t_i equal to the portion of paths where time t is used for task i: $\gamma'^t_i = \frac{\sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\hat{a}_{ik}^{pt}\lambda_k^p}{\sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\sum_{t'\in T_i}\hat{a}_{ik}^{pt'}\lambda_k^p}$. Using (19), (13) is satisfied since $\forall i\in C, \forall t\in T_i : r_i\gamma'^t_i = r_i\frac{\sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\hat{a}_{ik}^{pt}\lambda_k^p}{\sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\sum_{t'\in T_i}\hat{a}_{ik}^{pt'}\lambda_k^p} = \sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\hat{a}_{ik}^{pt}\lambda_k^p\frac{r_i}{\sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\sum_{t'\in T_i}\hat{a}_{ik}^{pt'}\lambda_k^p} \ge \sum_{k\in V}\sum_{p\in\mathcal{P}'_k}\hat{a}_{ik}^{pt}\lambda_k^p$. γ' obviously satisfies (14) and (15) is identical to (20). So for each solution to (18)-(21) there is a corresponding solution to the LP-relaxation of (12)-(17). Since

the objective functions (12) and (18) are identical, the projection on the λ-subspace of the LP-relaxation of (12)-(17) is a relaxation of (18)-(21). ■

Hence, instead of using the model directly, we relax the constraint on synchronized cooperation by using the model (18)-(21). We define $a_{ik}^p = \sum_{\forall t \in T_i} \hat{a}_{ik}^{pt}, \forall i \in C, \forall k \in V, \forall p \in \mathcal{P}_k$, where $a_{ik}^p = 1$ if task i is in path p for vehicle k and $a_{ik}^p = 0$ otherwise. At the same time, we choose to change from a maximization problem to a minimization problem by introducing δ_i as the number of unassigned split tasks of task i. This is our *relaxed master problem*. Finally, to decrease the size of the problem, a set of promising paths $\mathcal{P}'_k$ ($\subseteq \mathcal{P}_k$) is used instead of $\mathcal{P}_k$. In a column generation context $\mathcal{P}'_k$ contains all paths generated for team k in the pricing problem so far. We arrive at the *restricted master problem (RMP)*:

$$\min \sum_{i \in C} \delta_i \tag{22}$$

$$\delta_i + \sum_{k \in V} \sum_{p \in \mathcal{P}'_k} a_{ik}^p \lambda_k^p \geq r_i \qquad \forall i \in C \tag{23}$$

$$\sum_{p \in \mathcal{P}'_k} \lambda_k^p = 1 \qquad \forall k \in V \tag{24}$$

$$\lambda_k^p \geq 0 \qquad \forall k \in V, \forall p \in \mathcal{P}'_k \tag{25}$$

$$\delta_i \geq 0 \qquad \forall i \in C \tag{26}$$

The sum of δ_i over all tasks is minimized (22). (19) is changed to a greater-than inequality constraint, penalizing inadequate assignment to a task by adding δ_i (23). This change allows tasks to be done more times than required, which is useful in a column generation setting, where an existing column may enter the solution basis, and we do not have to generate a new, almost identical column containing a subset of the tasks. As a consequence, the estimates of the final dual variables improve (see [20]). The new master problem has the form of a generalized set-covering problem.

On the downside, any solution may now contain *overcovering*, i.e. we may have tasks which are assigned to more teams than requested. However, in the new formulation, overcovering can be removed without altering the objective value by unassigning the superfluous number of teams for each task. The modified solution is still feasible and the overcovering can hence easily be removed from an optimal solution.

If the master problem contains no columns representing paths from the outset of the column generation procedure, the problem will be infeasible due to the team constraints (24). Therefore, we add an *empty path* λ_k^0 ($a_{ik}^0 = 0, \forall i \in C$) for each team to ensure feasibility whether regular paths are present or not. An empty path can only be part of an optimal solution if the presence of the team can not decrease the number of unassigned tasks. This will be the case if manpower is available in abundance or the skills or working hours of the team do not match those of the tasks.

The solution to the restricted master problem may not be integer. In addition, we have relaxed the constraint on synchronization of tasks. Both of these properties must be enforced by a branching scheme.

The solution to the restricted master problem is not guaranteed to be optimal either, since only a small subset of feasible paths is considered. For each primal solution λ to the restricted master problem we obtain a dual solution $[\pi, \tau]$, where π and τ are the dual variables of constraints (23) and (24) respectively. In column generation, the dual solution is used in the pricing problem to ensure the generation of columns leading to an improvement of the solution to the master problem.

3.2 Pricing problem

The *pricing problem* specifies all the requirements of a feasible path. The objective is to find the path with the lowest possible cost. In m-MAPTWTC with inhomogeneous teams as described above, we obtain $m = |V|$ separate pricing problems. Each pricing problem is an *Elementary Shortest Path Problem with Time Windows (ESPPTW)*. The binary variable x_{ij} is defined as $x_{ij} = 1$ if the team goes directly from task i to task j and $x_{ij} = 0$ otherwise. For a team $k' \in V$ the pricing problem is formulated as:

$$\min \sum_{i \in C} \sum_{j \in C} -\pi_i x_{ij} - \tau_{k'} \tag{27}$$

$$\sum_{j \in N} x_{0j} = 1 \tag{28}$$

$$\sum_{i \in N} x_{ih} - \sum_{j \in N} x_{hj} = 0 \qquad \forall h \in N \tag{29}$$

$$e_{k'} + t_{0j} - M(1 - x_{0j}) \leq s_j \qquad \forall j \in C \tag{30}$$

$$s_i + t_{i0} - M(1 - x_{i0}) \leq f_{k'} \qquad \forall i \in C \tag{31}$$

$$s_i + t_{ij} - M(1 - x_{ij}) \leq s_j \qquad \forall i \in C, \forall j \in C \tag{32}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in C \tag{33}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in N, \forall j \in N \tag{34}$$

$$s_i \in \mathbb{Z}^+ \cup \{0\} \qquad \forall i \in C \tag{35}$$

The constraints match the constraints of the original formulation except for the relation between vehicles.

The pricing problem can be perceived as a graph problem. Consider a graph $G(N_G, E_G, c, t)$, where the nodes N_G are all tasks plus the service center and E_G is the set of edges connecting all nodes. With each edge $e \in E_G$ is associated a travel time $t_e = t_{ij}$ and a cost $c_e = c_{ij} = -\pi_i$, where i and j are the two nodes connected by e. To simplify, the service center is usually split into two vertices: a start vertex 0 and an end vertex $n + 1$. The objective is to find a path in G from 0 to $n + 1$ with a minimum sum of edge costs that does not violate any time windows.

Solution methods to the Shortest Path Problem with Time Windows have been studied extensively in the literature and successful algorithms for solving SPPTW have been built on the concept of dynamic algorithms. We solve the elementary version of the problem (ESPPTW), where no cycles are allowed. Dror [9] proves that the problem is NP-hard in the strong sense and thus no pseudo-polynomial algorithms are likely to exist. We use a label setting algorithm built on the ideas of Chabrier [3] and Jepsen et al. [18]. The authors of both papers have recently succeeded in solving previously unsolved VRPTW benchmarking instances (from the Solomon Test-sets [31]) by ESPPTW-based column generation. Furthermore, Feillet et al. [11], [12] address the Vehicle Routing Problem with Profits (similar to the Vehicle Routing Problem with a limited number of vehicles) and state that solving the elementary shortest path problem as opposed to the relaxed version is essential to obtain good bounds.

We will not go into the details of the label setting algorithm, since the problem is almost identical to the pricing problem of VRPTW. We have a shortest path problem where all arc costs out of a node are identical and hence can be moved to the node. The pricing problems are first solved in a heuristic label setting approach and if no columns can be added, we switch to the exact label setting algorithm.

3.3 Linking the Pricing Problem to the Master Problem

3.3.1 Team Priorities

As described earlier, each team has its separate pricing problem. This introduces the challenge of choosing the pricing problem in each iteration that is most likely to return usable columns. Initially, we implement a round-robin style mechanism, where each team is picked in turn. If a whole round is completed without at least one pricing problem returning a path with negative reduced cost, optimality is proven for the relaxed master problem.

Typically, some teams have less tight schedules than others and good columns are generated earlier in the process. We introduce another scheme to utilize this feature. We associate each team with a *team priority,* which is set equal to the reduced cost of the latest returned column. If no column was returned for team k, the team priority is set to a positive number higher than all other priority values to ensure that all other teams are treated before considering team k again.

By using team priorities, the teams which have recently shown the biggest improvements are treated first. Notice, that in some iterations we may not find the column with minimum reduced cost as it may be associated with a different team. However, when terminating the column generation, optimality is guaranteed in the same way as for the simple round-robin scheme.

3.3.2 Store Last Solved Pricing Problem

Having a number of separate teams with different skills and scheduling horizons means that the pricing problems of some teams do not change for many iterations. In the extreme case, we sometimes see master problems which are actually separable, i.e. the assignment of tasks to one team has no way of altering the dual variables for the pricing problem of another team. In these cases we may solve the exact same pricing problem repeatedly. To avoid this, we save the last solved pricing problem for each team, if it did not return any columns with negative reduced cost. If it did return such a column, there is no point in saving the problem as the dual variables will now have changed.

Prior to solving a pricing problem, it is checked whether any circumstances have changed since last time. These circumstances include dual variables and relevant branching decisions.

4 Branching

4.1 Branching to get integral solutions

Various branching strategies for VRPTW have been proposed. See [20] for a more thorough review of branching strategies for VRPTW. In the MAPTWTC setting, a 0-1 branching on an original flow variable x_{ijk} (proposed independently by Halse [15] and Desrochers et al. [8]) is equivalent to forcing team k to do (banning team k from doing, respectively) task j immediately after task i. The branching is enforced by removing illegal columns in the master problem in each child node and removing illegal arcs in the network formulation of the pricing problem for team k. In VRPTW, another possibility is to perform a 0-1 branching on $\sum_k x_{ijk}$ thus imposing the above constraint on all teams simultaneously. However, since the teams are inhomogeneous due to different qualifications and work hours and since tasks i and j may need several teams to cooperate, the branching rule is no longer a 0-1 branching and the advantage of keeping just one identical pricing problem for all teams is obviously lost.

Instead, we focus on a 0-1 branching scheme based on $\sum_j x_{ijk}$ which simply implies that team k is either forced to or banned from completing task i. Unlike the two strategies above, there is no need to keep track of the status of individual arcs in the pricing problems of the child nodes. The node corresponding to task i is either removed from the network (along with *all* arcs incident to it) or given a very low (negative) cost to ensure its inclusion in any optimal solution to the pricing problem.

4.2 Synchronized Cooperation using branching

Consider an optimal solution to the relaxed master problem, fractional or integral, and let s_i^p be the point in time where execution of task i begins on path p (if i is not a part of p, s_i^p is irrelevant). The solution violates the synchronized cooperation constraint for some task i if there exist positive variables $\lambda_{k_1}^{p_1}$ and $\lambda_{k_2}^{p_2}$ associated with the two paths p_1 and p_2 ($p_1 \neq p_2$), both containing i where

$$s_i^{p_1} \neq s_i^{p_2}$$

If the solution is fractional, the teams k_1 and k_2 may be identical. In this case, the team can be perceived as cooperating with itself.

Define $s_i^* = \lceil (s_i^{p_1} + s_i^{p_2})/2 \rceil$ as the *split time*. Now, split the problem into two branches and define new time windows for task i as

$$[a_i; s_i^* - 1] \qquad \text{and} \qquad [s_i^*; b_i]$$

respectively. Existing columns not satisfying the new time windows are removed from the corresponding child nodes and new columns generated must also respect the updated time window. In this way, the current solution is cut off in both branches and the new subspaces are disjoint. Since time has been discretized the branching strategy is guaranteed to be complete.

The idea behind this branching scheme is to restrict the number of points in time, where the execution of task i can begin. If the limited time window makes it inconvenient for the teams to complete task i, the lower bound will increase and the branch is likely to be pruned at an early stage. On the other hand, if the limited time window contains an optimal point in time for the execution of task i, it may be necessary to continue the time window branching until a singleton interval is reached. The time is discretized into a finite number of steps (minutes), and hence this will always be possible. However, since the label setting algorithm for the pricing problem aims at placing tasks as early as possible (see [8]), the actual number of different positions in time for any task is rather small. In fact, as the time windows are reduced, the tasks are more and more likely to be placed at the very beginning of their time window. This property greatly reduces the number of branching steps needed.

Using time window branching, the solution will eventually become feasible with respect to the synchronized cooperation constraint. It is not guaranteed to be integral, though, and it may therefore be necessary to apply the regular $\sum_j x_{ijk}$ branching scheme, branching on a combination of a task and a team. As both schemes have a finite number of branching candidates, the solution algorithm will terminate when they are used in combination. In general, when none of the feasibility criteria (integrality and synchronized cooperation) are fulfilled, we have a choice of branching scheme.

Our algorithm has been set to use time window branching whenever applicable. The restricted time windows reduce flexibility in the column generation which, in turn, limits the possibilities of combining fractional columns when solving the master problem. Thus, time window branching is also expected to have a positive influence on the integrality of

the solution as observed by Gélinas et al. [14] for VRPTW. This property has also been observed in practice when testing the algorithm, hence the choice of prioritizing time window branching.

We now focus on how good branching candidates are selected for branching. Let P_i be the set of all paths p including task i with $\lambda_k^p > 0$ in the current solution to the restricted master problem. If

$$s_i^{p_1} \neq s_i^{p_2}$$

for any two paths $p_1, p_2 \in P_i$, task i is stored in the set C' of possible candidates. We determine the split time as

$$s_i^* = \left\lceil \frac{\min_{p \in P_i}(s_i^p) + \max_{p \in P_i}(s_i^p)}{2} \right\rceil, \forall i \in C'$$

When ranking the branching candidates, we prefer candidates that provide a balanced search tree. That is, the paths in P_i should be divided equally into the two child nodes when weighted according to the variable values λ_k^p. Define

$$S_i = \sum_{k \in V, p \in P_i} \lambda_k^p, \forall i \in C'$$

as the sum of all positive variables containing i and let

$$S_i^< = \sum_{p \in P_i | s_i^p < s_i^*, k \in V} \lambda_k^p, \forall i \in C'$$

be the same sum restricted to the variables where task i is executed before the split time. The branching candidate i^* is now determined by

$$i^* = \arg\min_{i \in C'} \left| \frac{S_i^<}{S_i} - 0.5 \right|$$

5 Computational Results

The Branch-and-Price algorithm has been implemented in the Branch-and-Cut-and-Price framework of COIN-OR [4] and tests have been run on 2.7 GHz AMD processors with 2 GB RAM. The implementation has been tuned to the problems at hand and parameter settings have been made on the basis of these problems. The algorithm is set to do strong branching [1] with 25 branching candidates and adds up to 10 columns with negative reduced cost per pricing problem.

The test data sets originate from real-life situations faced by ground handling companies in two of Europe's major airports. This gives rise to four different problem types, since the two airports each produce problems of two distinctive types. Each type is represented by three problem instances, each spanning approximately one 24-hour day, thus, a total of 12 test instances are available.

Generally, the four problem types can be summarized as (In brackets: The total number of tasks after splitting into requested split tasks):

Type A Small instances, Airport 1. 12-13 teams and 80 (120) tasks

Type B Medium instances, Airport 2. 27 teams and 90 (150) tasks.

Type C Small instances, Airport 2. 15 teams and 90 (110) tasks.

Type D Large instances, Airport 1. 19-20 teams and 270 (300) tasks.

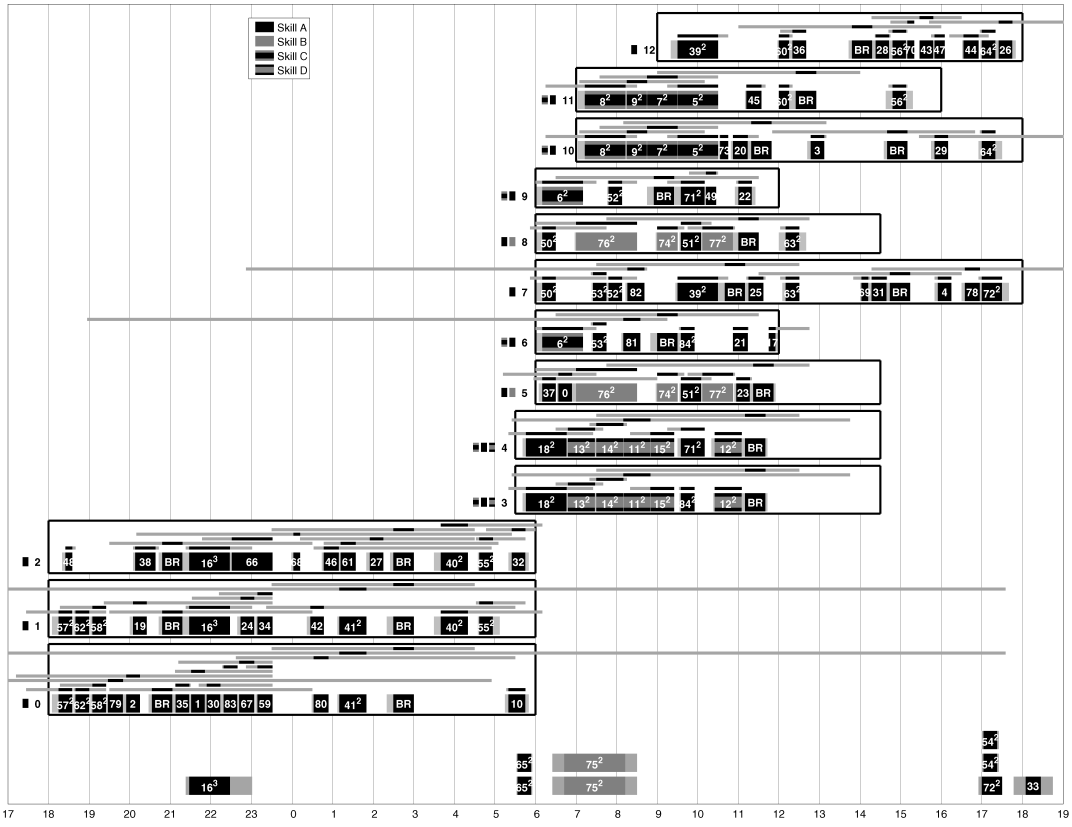

Figure 1: Problem instance **A.1** and its optimal solution.

The problem instance **A.1** and its optimal solution is illustrated in Figure 1. The figure depicts the distribution of tasks over the day and the skill requirements for these. The execution time of tasks and the length of their time windows are similar in the other problem types. In our problem instances, each team must be given a predefined number of breaks during their day and within certain time windows. Breaks are treated as regular tasks, with the exceptions that they can only be assigned to the related team, and they cannot be left unassigned in a feasible solution.

The individual schedules of the teams are captured in the 13 boxes, which clearly show the start and end time of each shift. Each task is represented by one or more small boxes labeled with the task ID (Breaks have ID: "BR"). The superscript denotes the number of teams that the task must be split between. This number therefore corresponds to the total number of boxes labeled with the task ID of this task. Above each task is a thin box depicting the time window of the task. Furthermore, each task has a color pattern revealing its skill requirement. Each team has between one and three skills, identified by the small squares to the left of the team ID. To assign a task to a team, the color pattern of the task must match that of one of these small squares.

To illustrate how to read the figure, we go through the work plan of team 9. The first task carried out is task 6 which requires skill C. The task is scheduled from 6:10 to 7:10 and hence the time window of the task is respected, since execution cannot start before 6 o'clock and must be finished by 7:30. The task is solved in collaboration with team 6. The light gray box in front of the task gives the required travel time. Next, the team takes care of task 52 (requires skill A), this time cooperating with team 7. After this, team 9 is given their daily break. Subsequently, they will carry out 71, 49, and 22, where task 49 and task 22 are dealt with by team 9 alone.

	A.1	**A.2**	**A.3**	**B.1**	**B.2**	**B.3**	**C.1**	**C.2**	**C.3**	**D.1**	**D.2**	**D.3**
Unassigned split tasks	9	*7	1	0	3	5	*3	*6	*10	*29	24	*31
Lower Bound$^{\otimes}$	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)	133	OM	2663	120	172	97	OM	OM	OM	TO	2719	TO
- LP (%)	15	46	20	10	10	11	29	9	34	2	5	3
- Branching (%)	68	7	70	82	82	78	34	81	32	5	10	4
- Pricing Problem (%)	4	8	2	1	2	2	4	4	9	93	83	91
- Overhead (%)	13	39	8	7	6	9	33	6	25	0	2	2
Tree size	605	42435	3207	537	597	507	188623	87843	69637	4961	487	2741
Max. depth	160	162	168	264	291	253	122	166	204	219	235	228
# Pricing Problems	13292	$3 \cdot 10^6$	107320	15554	17240	14813	$3 \cdot 10^6$	$2 \cdot 10^6$	$2 \cdot 10^6$	379799	20728	247634
# Vars added	12268	$2 \cdot 10^6$	109810	4074	5223	4321	$2 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	231209	16659	204614

Table 1: Results of the Branch-and-Price algorithm with no initial solution.
OM = Out-of-Memory was encountered. TO = The Time-Out limit of 10 hours was reached.
* The solution given is the best feasible solution found.
$^{\otimes}$ Lower Bound (more details in Table 3).

In Table 1 the results from the 12 datasets are given. From the table we conclude the following. 6 of the 12 datasets were solved to optimality within one hour. The remaining 6 instances are split in two cases: one case for the small and medium-sized problems (**Type A-C)** and one case for the large instances (**Type D)**. For the unsolved problems of **Type A-C** we see an explosion in the size of the branching tree. In these cases the time-out limit is never reached, since we run out of memory before time out. The reported results for these instances have been recorded after 2 hours, which in these cases is just before the memory limit is reached. For **Type D** the results indicate that the generation of columns is now in itself a time-consuming task and time-out is encountered with a relatively small tree-size.

The branching trees from the above test have been built without a good initial solution. For each of the unfinished problems, we restart the algorithm with an initial solution, namely the best feasible solution of Table 1. The results of the new test are displayed in Table 2.

It is interesting that most of these instances are now solved to optimality within seconds. It clearly indicates that inexpedient branching decisions were made in the first run and more reliable branching is possible when promising columns exist initially. Another observation is that solving **C.1** under default settings leads to another out-of-memory failure, whereas changing the settings slightly gives an optimal solution within one second. This is another indication of the importance of making the right branching decisions and the consequence of not doing so. It has been tested that the settings giving a fast solution in this case are not superior in general.

Systematic exploitation of these features is outside the scope of this article. Automatic restart of the branching procedure could be implemented fairly easy. To achieve even faster

	A.1	A.2	A.3	B.1	B.2	B.3	C.1	C.2	C.3	D.1	D.2	D.3
Unassigned split tasks	9	7	1	0	3	5	$^{\times}3$	4	9	$^{*}29$	24	31
Lower Bound$^{\otimes}$	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)		0.84					0.80	36	0.97	TO		235
- LP (%)		33					25	21	17	0		5
- Branching (%)		5					8	25	8	0		0
- Pricing Problem (%)		18					6	14	8	100		95
- Overhead (%)		44					61	40	67	0		0
Tree size		11					19	981	59	447		9
Max. depth		3					5	46	28	40		4
# Pricing Problems		530					561	32921	1358	42284		6415
# Vars added		785					758	16406	475	37212		6104

Table 2: Results of the Branch-and-Price algorithm with initial solution from the test of Table 1.
TO = The Time-Out limit of 10 hours was reached.
* The solution given is the best feasible solution found.
$^{\times}$ After OM on the first run, the pricing problem solver was in this case changed to not create heuristic columns.
$^{\otimes}$ Lower Bound (more details in Table 3).

results, a variety of acceleration strategies should be investigated. Look to [6] for more on this topic.

To reveal the complexity added by the synchronized cooperation requirement, we also show results for a version of the problem where no branching on time windows is done (Table 3). This means that cooperation is no longer synchronized, but we are able to reach optimal solutions faster. Since the latter is a relaxation of the original problem, we are able to use the solution values as lower bounds on our problem.

	A.1	A.2	A.3	B.1	B.2	B.3	C.1	C.2	C.3	D.1	D.2	D.3
Unassigned split tasks	9	6	1	0	3	5	2	4	9	27	24	30
Time (s)	0.96	1.10	1.37	0.64	0.77	0.80	1.18	1.86	1.65	75.18	413.26	2195.59
- LP (%)	16	8	15	6	4	3	19	25	10	17	10	2
- Branching (%)	7	0	0	0	0	1	39	24	49	17	8	1
- Pricing Problem (%)	45	74	69	19	22	30	9	11	17	62	81	97
- Overhead (%)	32	18	16	75	74	67	33	40	24	4	1	0
Tree size	3	3	1	1	3	3	11	21	19	35	83	97
Max. depth	1	1	0	0	1	1	5	8	9	17	41	22
# Pricing Problems	163	93	291	103	80	81	288	481	367	4450	8783	9811
# Vars added	407	350	663	309	222	212	586	683	435	4489	7773	14111

Table 3: Results of the Branch-and-Price algorithm with no constraint on synchronized coordination.
All solution values can be used as lower bounds on the original formulation.

Solution times of Table 3 should be compared to the times of Table 1 and reveal that solving the relaxed problem evidently is much faster and optimal solutions are found in all cases. The running times for the small and medium problems are up to 2 seconds, where one

of the large problem instances uses around 37 minutes.

It is conspicuous that all the optimal solutions found in Table 1 are equal to the lower bound found in Table 3. The lower bound found by the unsynchronized model is naturally closely related to the lower bound found in the root node of the branching tree of the problems in Table 1 and these results stress how important a good lower bound is.

6 Conclusion and future work

The Manpower Allocation Problem with Time Windows, Job-Teaming Constraints and a limited number of teams is successfully solved to optimality using a Branch-and-Price approach. By relaxing the synchronization constraint and using Dantzig-Wolfe decomposition, the problem is divided into a generalized set covering master problem and an elementary shortest path pricing problem. Applying branching rules to enforce integrality as well as synchronized execution of divided tasks enables us to arrive at optimal solutions in half of the test instances. Running a second round of the optimization, initiated from the best solution found in round one, uncovers the optimal solution to all but one of the 12 test instances. The test instances are all full-size realistic problems originating from scheduling problems of ground handling tasks in major airports. Synchronization between teams in an exact optimization context has not previously been treated in the literature. We have successfully integrated the extra requirements into the solution procedure and the results are promising.

Future work could aim at creating a structured approach to utilize the effect of restarting the branching mechanism. By simply restarting the algorithm once, we see a remarkable increase in the number of solvable problems, and an extended strategy may shorten solution time significantly and it may further increase the chance of finding optimal solutions. Other acceleration strategies are likely to reveal improved results as well.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] J.E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[3] A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers and Operations Research*, 33(10):2972–2990, 2006.

[4] Coin. COmputational INfrastructure for Operations Research (COIN-OR), 2006. http://www.coin-or.org/.

[5] W. Cook and J.L. Rich. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical report, Rice University, Houston, TX, USA, 1999.

[6] E. Danna and C.L. Pape. *Branch-and-Price Heuristics: A Case Study on the Vehicle Routing Problem with Time Windows*, chapter 4, pages 99–129. Desaulniers G., Desrosiers J., Solomon M.M.: Column Generation, Springer, New York, 2005.

[7] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[8] M. Desrochers, J. Desrosiers, and M.M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

[9] M. Dror. Note on the complexity of the shortest path models for column generation in VRPTW. *Operation Research*, 42(5):977–978, 1994.

[10] M. Dror and P. Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–149, 1989.

[11] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[12] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[13] P. W. Frizzell and J. W. Giffin. The split delivery vehicle scheduling problem with time windows and grid network distances. *Computers and Operations Research*, 22(6):655–667, 1995.

[14] S. Gélinas, M. Desrochers, J. Desrosiers, and M.M. Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

[15] K. Halse. *Modeling and Solving Complex Vehicle Routing Problems*. PhD thesis, Technical University of Denmark, 1992.

[16] S. C. Ho and D. Haugland. A tabu search heuristic for the vehicle routing problem with time windows and split deliveries. *Computers and Operations Research*, 31(12):1947–1964, 2004.

[17] S. Irnich and D. Villeneuve. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3), 2006.

[18] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. A non-robust branch-and-cut-and-price algorithm for the vehicle routing problem with time windows. Technical report, Department of Computer Science, University of Copenhagen, Denmark, 2006.

[19] B. Kallehauge, J. Larsen, and O.B.G. Madsen. Lagrangean duality applied on vehicle routing with time windows - experimental results. Technical report, IMM, Technical University of Denmark, Copenhagen, Denmark, 2001.

[20] B. Kallehauge, J. Larsen, O.B.G. Madsen, and M.M. Solomon. *Vehicle Routing Problem with Time Windows*, chapter 3, pages 67–98. Desaulniers G., Desrosiers J., Solomon M.M.: Column Generation, Springer, New York, 2005.

[21] N. Kohl. *Exact Methods for Time Constrained Routing and Related Scheduling Problems*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Denmark, 1995.

[22] N. Kohl, J. Desrosiers, O.B.G. Madsen, M.M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[23] Jesper Larsen. *Parallelization of the Vehicle Routing Problem With Time Windows*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Denmark, 1999.

[24] H.C. Lau, M. Sim, and K.M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148:559–569, 2003.

[25] Chi-Guhn Lee, Marina A. Epelman, Chelsea C. White III, and Yavuz A. Bozer. A shortest path approach to the multiple-vehicle routing problem with split pick-ups. *Transportation Research Part B*, 40:265–284, 2006.

[26] Y. Li, A. Lim, and B. Rodrigues. Manpower allocation with time windows and jobteaming constraints. *Naval Research Logistics*, 52:302–311, 2005.

[27] Zhiye Li, Songshan Guo, Fan Wang, and Andrew Lim. Improved GRASP with tabu search for vehicle routing with both time window and limited number of vehicles. In *Innovations in Applied Artificial Intelligence*, pages 552–561, 2004. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004.

[28] A. Lim, B. Rodrigues, and L. Song. Manpower allocation with time windows. *Journal of the Operational Research Society*, 55:1178–1186, 2004.

[29] A. Lim and Xingwen Zhang. A two-stage heuristic for the vehicle routing problem with time windows and a limited number of vehicles. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 3 - Volume 03*. IEEE Computer Society, 2005.

[30] G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

[31] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[32] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, Inc., 1998.